

EDK Build Specification

May 2010
Revision 1.22

Acknowledgements

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

A license is hereby granted to copy and reproduce this specification for internal use only.

No other license, express or implied, by estoppel or otherwise, to any other intellectual property rights is granted herein.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

This specification is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

*Other names and brands may be claimed as the property of others.

Copyright © 2008 - 2010 Intel Corporation. All rights reserved.

Revision History

Revision	Revision History	Date
1.0	Initial release .	February 2008
1.1	Update based on errata	August 2008
1.2	Updates based on enhancement requests	June 2009
1.21	Updates based on errata and enhancement requests • Definitions in DSC file [defines] section are now global to both DSC and FDF files • Added language filters: RFC_LANGUAGES and ISO_LANGUAGES • Rule processing for file type lists is alphabetical, i.e., files are added in alphabetical order • Added warning for VFR file naming convention - cannot use a name that is also used for a C file • Use of the IDENTIFIER statement in tools_def.txt is optional • Whitespace characters are permitted in the meta-data files, so tools must handle them (use of token based processing is recommended) • Tools must support any number of FV_EXT_ENTRY_TYPE statements in an FDF file • The build tools must auto compute the ExtHeaderOffset in the FV Header. • The build tools must auto compute the ExtHeaderSize in the FV Ext Header based on the size of the FV Ext Header and all the FV Ext Header Entries. • The build tools must auto compute the ExtEntrySize field in the FV Ext Header Entry structure based on the size of the file specified by the FILE statement or the number of bytes in the byte array of the DATA statement. If the size is greater than 16-bits, the build should break. • Specified parsing priority rules for definitions and values • Added report generator syntax as part of the build • Add support for “Auto” alignment for PE32 and TE images • Add support for specifying block information for capsules • FeatureFlagExpression processing should allow C-style expression syntax and follow C rules for processing	January 2010
1.22	Grammatical and formatting changes. Added Module types appendix.	May 2010

Contents

1

Introduction 1

1.1 Overview 1
1.2 Target Audience 1
1.3 Related Information 1
1.4 Terms 2
1.5 Conventions Used in this Document 5
1.5.1 Data Structure Descriptions 5
1.5.2 Pseudo-Code Conventions 6
1.5.3 Typographic Conventions 6

2

Design Discussion 9

2.1 Reference Implementation: UEFI & PI formats 9
2.2 Development Environments 9
2.2.1 Reference Implementation: Build process restrictions 9
2.3 UEFI/PI Firmware Images 9
2.4 Boot Sequence 11
2.4.1 Security (SEC) 12
2.4.2 Pre-EFI Initialization (PEI) 12
2.4.3 Drive Execution Environment (DXE) 13
2.4.4 Boot Device Selection (BDS) 13
2.4.5 Transient System Load (TLS) and Runtime (RT) 13
2.4.6 After Life (AL) 13
2.5 Typical Flash Part Layout 13
2.6 Generic Build Process 16
2.6.1 EFI SECTION Files 17
2.6.2 Firmware Files 18
2.6.3 Firmware Volumes 20
2.6.4 Special Files - VTF & BSF 22
2.6.5 EFI_FV_FILETYPE_SECURITY Notes 22
2.6.6 EFI_FV_FILETYPE_PEI_CORE Notes 22
2.6.7 EFI_FV_FILETYPE_PEIM Notes 23
2.6.8 EFI_FV_FILETYPE_COMBINED_PEIM_DRIVER Notes 23
2.6.9 DXE, BDS, TLS and AL Notes 23
2.7 Creating EFI Images 24
2.7.1 Compiling Code 24
2.7.2 Creating a Terse Image 24
2.7.3 Removing reloc sections 25
2.7.4 Generating LEAF EFI_SECTION Files 26
2.7.5 Generating Encapsulation EFI_SECTION Files 28
2.7.6 Generating DEPEX EFI_SECTION Files 28
2.7.7 Generating Visual Forms (IFR - HII) Files 29
2.7.8 Generating EFI FFS Files 30

2.7.9 APRIORI Files 31
2.7.10 Generating EFI Firmware Volume (FV) Files 31
2.7.11 Implementing Compression 33
2.7.12 Implementing Encryption or Signing 34
2.7.13 Generating an FD image file 34
2.7.14 Generating Applications 35
2.7.15 Generating an Option ROM file 35
2.7.16 Generating Capsule Update Files 37

3
UEFI and PI Image Specification 39

4
Reference Build Process Overview 63
4.1 Supported Development Environments 64
4.2 Supported Development Tools 64
4.3 GCC Setup 65
4.4 Build Specific Meta-Data Files 65
4.4.1 Parse Meta-Data - AutoGen stage 66
4.4.2 Build Binary EFI Images - $(MAKE) stage 68
4.4.3 Assemble FLASH Images - ImageGen stage 68
4.4.4 EFI PCI Expansion Option ROM Images 69
4.4.5 UEFI Applications 69
4.5 File Specifications 69
4.6 File Extensions 70
4.6.1 AutoGen Stage File Extensions 70
4.6.2 File Extensions for UEFI image files. 71
4.6.3 ImageGen File Extensions 72
4.7 Getting Started 73
4.7.1 Workstation Setup 73
4.7.2 Downloading Code 74
4.7.3 Path and Target Verification 74
4.7.4 Pre-build Setup 74
4.7.5 Building Nt32 Emulation Environment 75
4.7.6 Simple Build Environment Tests 76

5
MetaData File Specifications 77
5.1 Meta-Data File Format 77
5.1.1 Section Entries (build_rule.txt only) 77
5.1.2 Comments 77
5.1.3 Valid Entries 77
5.1.4 Macro statements (tools_def.txt only) 78
5.1.5 Conditional Directive Statements 78
5.2 tools_def.txt 79
5.3 target.txt File 86
5.4 build_rule.txt 88

6
Build Environment 95
6.1 Environment Variables 95
6.1.1 Required Environment Variables 95
6.1.2 Optional Environment Variables 96
6.2 Build Scoping 96
6.2.1 The precedence of what (platform or module) gets built 97
6.2.2 The precedence of the TARGET value 97
6.2.3 The precedence of the TARGET_ARCH values 97
6.2.4 Third Party tools using -t TOOL_CHAIN_TAG 98
6.2.5 Precedence of FLAGS values 98
6.2.6 Precedence of Macro Definitions 98
6.2.7 Precedence of PCD Values 99
6.3 Third Party Tool Flags 99

7
AutoGen Stage 105
7.1 Overview 105
7.2 Auto-generation Process 106
7.2.1 Determine What to Build 106
7.2.2 Parse file pointed to by TOOL_CHAIN_CONF 109
7.2.3 Parse build_rule.txt specified by BUILD_RULE_CONF and ADD_BUILD_RULE .. 109
7.2.4 Parse DSC, FDF, INF, DEC files 110
7.2.5 Post processing 110
7.3 Auto-generated code 111
7.3.1 Dependency expression file 111
7.3.2 HII String Pack 111
7.3.3 AutoGen.h file 113
7.3.4 AutoGen.c file 119
7.4 Auto-generated Makefiles 136
7.4.1 Module Makefile 136
7.4.2 Top level Makefile 142
7.5 Binary Modules 145

8
$(MAKE) Stage 147
8.1 Overview 147
8.2 Preprocess/Trim 148
8.3 Compile/Assembly 148
8.4 Static Link 149
8.5 Dynamic Link 149
8.6 Generate Module Images 149
8.6.1 GenFw 149
8.7 Generate Platform Images 149

9
ImageGen Stage - FLASH Images 151
9.1 Overview of Flash Device Layout 151
9.2 Parsing FDF Meta-Data File 152
9.2.1 FILE Format Example 153
9.3 Build Intermediate Images 153
9.3.1 Binary modules 153
9.3.2 Creating EFI Sections 153
9.3.3 Create an Apriori File 154
9.3.4 Create FFS Files from Leaf Sections 155
9.3.5 Create Encapsulation Sections 156
9.4 Create the FV Image File(s) 157
9.5 Create the FD image file(s) 159
9.5.1 FV Region Type 160
9.5.2 DATA Region Type 160
9.5.3 FILE Region Type 161

10
ImageGen Stage - Other Images 163
10.1 EFI PCI OptionROM Images 163
10.2 UEFI Applications 163
10.3 Capsules 163

11
Build Changes and Customizations 165
11.1 Building for Debug 165
11.1.1 Debugging Files 165
11.1.2 Debugging Options 166
11.1.3 Advanced Debugging 166
11.2 Adding Custom Compression Tools 166
11.3 Using Custom Build Tools 167
11.4 Customizing Compilation for a Component 167
11.5 Changing Files in a Library [EDK I Only] 167

12
Reports 169
12.1 Build Report Generation (BRG) Tool 169
12.2 Sample Launch Steps: NT32 platform 169
12.3 Output 169
12.3.1 Layout 170
12.3.2 Section and Sub-section Format 170
12.4 Platform Summary 171
12.5 Global PCD Section 172
12.5.1 Required line 172
12.5.2 Optional lines 172
12.6 FD Section 173
12.6.1 FD Section Header 173
12.6.2 FD Region Sub-section 174

12.7 Module Section 175
12.7.1 Module Section Summary 175
12.7.2 Library Sub-section 176
12.7.3 PCD Sub-section 178
12.7.4 DEPEX Sub-section 179
12.7.5 Build Flags Sub-section 179
12.7.6 Notification Sub-section 180
Appendix A
Variables 181
Appendix B
tools_def.txt 185
Appendix C
target.txt 255
Appendix D
build_rule.txt 259
Appendix E
build.exe command 269
E.1 Overview 269
E.2 Makefile actions 269
E.3 Build Targets and options 269
E.4 Usage 270
Appendix F
NT32 Platform Emulation Envoronment 273
Appendix G
Firmware Volume INF 275
G.1 Firmware Volume INF Description 275
G.2 [Attributes] Section 275
G.3 [Files] Section 277
G.4 [Options] Section 278
Appendix H
HII - EFI Compliant Unicode Strings File Format 281
H.1 Control Character 281
H.1.1 Language Definition 281
H.2 String Definitions 281
H.3 String Control Sequences 282
Appendix I
VS2005 Team Suite Performance Profile 285
I.1 Step 1 - Create a new project 285
I.2 Step 2 - Update the project 285
I.2.1 To pass an argument in to the console application 287
I.2.2 Step 3 Run the Performance Wizard 289
Appendix J

HII - UEFI Compliant Unicode Strings File Format 295
Appendix K
Module Types 299

Tables

Table 1.EDK Build Infrastructure Support Matrix 1
Table 2. EFI Section Types 18
Table 3. Defined FV File Types 20
Table 4. Basic EFI_SECTION Type Codes 27
Table 5. Encapsulation EFI_SECTION Type Codes 28
Table 6. Dependency Section Type Codes 29
Table 7. VFR Compatibility Matrix 30
Table 8. Descriptions of Bit Fields for Attributes 53
Table 9. Supported Compiler Tools 64
Table 10. Additional Compiler Tools 65
Table 11. AutoGen Stage Input File Extensions 71
Table 12. $(MAKE) Stage Intermediate Output File Extensions 71
Table 13. $(MAKE) Stage Output File Extensions 72
Table 14. GenFds Image Generation: Intermediate File Extensions 73
Table 15. ImageGen Final Output File Extensions 73
Table 16. Predefined Command Codes 83
Table 17. Predefined Attributes 84
Table 18. Reserved Variable Strings 91
Table 19. Valid Macro Values 92
Table 20. Standard C File Compiler Options 100
Table 21. Assembly Flags 101
Table 22. C Compiler's Preprocessor Options 101
Table 23. C Compiler's Preprocessor Options for VFR files ONLY 101
Table 24. Pre-compiled Header (PCH) Creation Flags 101
Table 25. Static Linker Flags 103
Table 26. Dynamic Linker Flags 103
Table 27. Variable Descriptions 182
Table 28. Build Targets and Comand-line Options 270
Table 29. HII Double-Byte Encoding Map 298
Table 30. EDK II Module Types 299

Figures

Figure 1. UEFI/PI Firmware Image Creation 10
Figure 2. EFI PCI Expansion OptionROM and UEFI Application Creation 11
Figure 3. PI Firmware Phases 12
Figure 4. NT32 Flash Device Layout 14
Figure 5. Typical IA32/X64 Flash Device Layout 15
Figure 6. Typical IPF FD Layout 16
Figure 7. General EFI Section Format (< 16MB) 17
Figure 8. General EFI Section Format for Large Size Sections. 17
Figure 9. Typical FFS File Layout (<16MB) 19
Figure 10. File Header 2 layout for files larger than 16Mb 19
Figure 11. General FV Layout 21
Figure 12. Standard Image to Terse Image Comparison 25
Figure 13. EFI Image Files 27
Figure 14. Depex File 29
Figure 15. Firmware Volume Layout 33
Figure 16. EFI PCI Expansion Option ROM layout 36
Figure 17. EFI Capsule Layout 38
Figure 18. EDK II Build Process Flow 63
Figure 19. EDK II AutoGen Process 105
Figure 20. EDK II Build Process - Platform Point of View (PoV) 147
Figure 21. EDK II Build Process - Module PoV 148
Figure 22. FD image generation process 152
Figure 23. Capsule Creation Process. 164
Figure 24. VS2005 Property Page 288
Figure 25. VS2005 Performance Summary 292
Figure 26. VS2005 Call Tree View 293

1 Introduction

1.1 Overview

This document describes the EDK II Build Architecture. This specification was designed to support new build requirements for building EDK components within the EDK II build infrastructure.

EDK II Build utilities described in this document use INI style text based meta-data files to describe components, modules, libraries, platforms, firmware volumes and firmware device images.

This document describes the high level EDK II Build Architecture, which has the following goals:

Compatible

The EDK II build environment must maintain backward compatibility with the existing EDK files. This means that the changes made to this specification must not require changes to existing files.

Compatibility is maintained by providing the EDK Tools in the EDK Compatibility package. Also, some INF files may require modification for the EDK II build environment if they are used to access Flash firmware - the PI 1.0 specification modified the flash data structures and defined some new GUID values.

Simplified platform build and configuration

One goal of this format is to simplify the build setup and configuration for a given platform. It was also designed to simplify the process of adding EDK and EDK II firmware components to a firmware volume on a given platform.

Specification Conformance

The EDK II Build infrastructure supports building UEFI 2.3 and PI 1.2 compliant platforms. Existing EDK components may need to be updated to align with these specifications.

Table 1. EDK Build Infrastructure Support Matrix

	EDK DSC	EDK II DSC	EDK FDF	EDK II FDF	EDK INF	EDK II INF
EDK Build Tools	YES	NO	YES	NO	YES	NO
EDK II Build Tools	NO	YES	NO	YES	YES	YES

1.2 Target Audience

This document is intended for persons performing EFI development and support for different platforms.

1.3 Related Information

The following publications and sources of information may be useful to you or are

referred to by this specification:

- *Extensible Firmware Interface Specification*, Intel, 2001, http://developer.intel.com/technology/efi.

Note: *See the glossary sections in the EFI 1.10 Specification for additional required or recommended definitions of terms, abbreviations, documents and specifications:.*

Note: *See the master Framework glossary in the Framework Interoperability and Component Specifications help system for additional required or recommended definitions of terms, abbreviations, documents and specifications:* http://www.intel.com/technology/framework/spec.htm.

- *Unified Extensible Firmware Interface Specification*, Version 2.3, Unified EFI, Inc, 2006, http://www.uefi.org.
- *Platform Initialization Specification*, Version 1.1, Unified EFI, Inc., 2006, http://www.uefi.org.
- *Intel® Platform Innovation Framework for EFI Specifications*, Intel, 2006, http://www.intel.com/technology/framework/.
- http://sourceforge.net/projects/edk2/files/
  - *EDK II Module Writers Guide,* Intel, 2010,.
  - *PCD Infrastructure*, Intel, 2009,.
  - *EDK II User Manual*, Intel, 2010.
  - *EDK II C Coding Standard*, Intel, 2010.
  - *EDK II DSC Specification*, Intel, 2010.
  - *EDK II INF File Specification*, Intel, 2010.
  - *EDK II FDF Specification*, Intel, 2010.
  - *EDK II Build Specification, Intel*, 2010.
  - *VFR Programming Language*, Intel, 2010.
  - *EDK II MDE Library Specification,* Intel, 2007

1.4 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

BaseTools

The BaseTools is a separate project from EDK II which supports the tools required for an EDK II build. This project is hosted at http://sourceforge.net/projects/edk2-buildtools/files/.

BDS

Framework Boot Device Selection phase.

BNF

BNF is an acronym for "Backus Naur Form." John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language.

Component

An executable image. Components defined in this specification support one of the defined module types.

DXE

Framework Driver Execution Environment phase.

DXE SAL

A special class of DXE module that produces SAL Runtime Services. DXE SAL modules differ from DXE Runtime modules in that the DXE Runtime modules support Virtual mode OS calls at OS runtime and DXE SAL modules support intermixing Virtual or Physical mode OS calls.

DXE SMM

A special class of DXE module that is loaded into the System Management Mode memory.

DXE Runtime

Special class of DXE module that provides Runtime Services

EBNF

Extended "Backus-Naur Form" meta-syntax notation with the following additional constructs: square brackets "[…]" surround optional items, suffix "*" for a sequence of zero or more of an item, suffix "+" for one or more of an item, suffix "?" for zero or one of an item, curly braces "{…}" enclosing a list of alternatives and super/subscripts indicating between n and m occurrences.

EFI

Generic term that refers to one of the versions of the EFI specification: EFI 1.02, EFI 1.10, UEFI 2.0, UEFI 2.1, UEFI 2.2 or UEFI 2.3.

EFI 1.10 Specification

Intel Corporation published the Extensible Firmware Interface Specification. Intel has donated the EFI specification to the Unified EFI Forum, and the UEFI now owns future updates of the EFI specification.

FLASH

This term is used throughout this document to describe one of the following:

- An image that is loaded into a hardware device on a platform - traditional ROM image
- An image that is loaded into an Option ROM device on an add-in card
- A bootable image that is installed on removable, bootable media, such as a Floppy, CD-ROM or USB storage device.
- An image that is contains update information that will be processed by OS Runtime services to interact with EFI Runtime services to update a traditional ROM image.
- A UEFI application that can be accessed during boot (at an EFI Shell Prompt), prior to hand-off to the OS Loader.

Foundation

The set of code and interfaces that glue implementations of EFI together.

Framework

Intel® Platform Innovation Framework for EFI consists of the Foundation, plus other modular components that characterize the portability surface for modular components designed to work on any implementation of the EFI architecture.

GUID

Globally Unique Identifier. A 128-bit value used to name entities uniquely. A unique GUID can be generated by an individual without the help of a centralized authority. This allows the generation of names that will never conflict, even among multiple, unrelated parties. GUID values can be registry format (8-4-4-4-12) or C data structure format.

HII

Human Interface Infrastructure. This generally refers to the database that contains string, font, and IFR information along with other pieces that use one of the database components.

IFR

Internal Forms Representation. This is the binary encoding that is used for the representation of user interface pages.

Library Class

A library class defines the API or interface set for a library. The consumer of the library is coded to the library class definition. Library classes are defined via a library class .h file that is published by a package.

Library Instance

An implementation of one or more library classes.

Module

A module is either an executable image or a library instance. For a list of module types supported by this package, see module type.

Module Type

All libraries and components belong to one of the following module types: BASE, SEC, PEI_CORE, PEIM, DXE_CORE, SMM_CORE, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SMM_DRIVER, DXE_SAL_DRIVER, UEFI_DRIVER, or UEFI_APPLICATION. These definitions provide a framework that is consistent with a similar set of requirements. A module that is of module type BASE, depends only on headers and libraries provided in the MDE, while a module that is of module type DXE_DRIVER depends on common DXE components. For a definition of the various module types, see Appendix K.

Module Surface Area (MSA)

The MSA is an XML description of how the module is coded. The MSA contains information about the different construction options for the module. After the module is constructed the MSA can describe the interoperability requirements of a module.

Package

A package is a container. It can hold a collection of files for any given set of modules. Packages may be described as one of the following types of modules:

- source modules, containing all source files and descriptions of a module
- binary modules, containing EFI Sections or a Framework File System and a description file specific to linking and binary editing of features and attributes specified in a Platform Configuration Database (PCD).
- mixed modules, with both binary and source modules

Multiple modules can be combined into a package, and multiple packages can be combined into a single package.

Protocol

An API named by a GUID.

PCD

Platform Configuration Database.

PEI

Pre-EFI Initialization Phase.

PPI

A PEIM-to-PEIM Interface that is named by a GUID.

Runtime Services

Interfaces that provide access to underlying platform-specific hardware that might be useful during OS runtime, such as time and date services. These services become active during the boot process but also persist after the OS loader terminates boot services.

SAL

System Abstraction Layer. A firmware interface specification used on Intel® Itanium® Processor based systems.

SEC

Security Phase is the code in the Framework that contains the processor reset vector and launches PEI. This phase is separate from PEI because some security schemes require ownership of the reset vector.

SKU

Stock Keeping Unit.

UEFI Application

An application that follows the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.

UEFI Driver

A driver that follows the UEFI specification.

UEFI Specification Version 2.3

Current version of the EFI specification released by the Unified EFI Forum.

UEFI Platform Initialization Specification 1.2

Current version of the PI specification released by the Unified EFI Forum. This specification builds on the Intel Platform Innovation Framework for EFI specifications and transfers ownership of most of these specifications from Intel to a non-profit, industry trade organization.

Unified EFI Forum

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see www.uefi.org.

VFR

Visual Forms Representation.

1.5 Conventions Used in this Document

This document uses typographic and illustrative conventions described below.

1.5.1 Data Structure Descriptions

Intel® processors based on 32 bit Intel® architecture (IA 32) are "little endian" machines. This distinction means that the low-order byte of a multibyte data item in memory is at the lowest address, while the high-order byte is at the highest address. Processors of the Intel® Itanium® processor family may be configured for both "little endian" and "big endian" operation. All implementations designed to conform to this specification will use "little endian" operation.

In some memory layout descriptions, certain fields are marked reserved. Software must initialize such fields to zero and ignore them when read. On an update operation, software must preserve any reserved field.

The data structures described in this document generally have the following format:

STRUCTURE NAME

The formal name of the data structure.

Summary:

A brief description of the data structure.

Prototype:

An EBNF-type declaration for the data structure..

Example:

Sample data structure using the prototype.

Description

A description of the functionality provided by the data structure, including any limitations and caveats of which the caller should be aware.

Related Definitions

The type declarations and constants that are used only by this data structure.

1.5.2 Pseudo-Code Conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a *list* is an unordered collection of homogeneous objects. A *queue* is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be FIFO.

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the *Extensible Firmware Specification*.

1.5.3 Typographic Conventions

This document uses the typographic and illustrative conventions described below:

Typographic Convention	Typographic convention description
Plain text	The normal text typeface is used for the vast majority of the descriptive text in a specification.
Plain text (blue)	Any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink.
Bold	In text, a **Bold** typeface identifies a processor register name. In other instances, a **Bold** typeface can be used as a running head within a paragraph.
Italic	In text, an *Italic* typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.

Typographic Convention	Typographic convention description
BOLD Monospace	Computer code, example code segments, and all prototype code segments use a **BOLD Monospace** typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.
Bold Monospace	Words in a **Bold Monospace** typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink.
$(VAR)	This symbol VAR defined by the utility or input files.
Italic Mono-space	In code or in text, words in *Italic Monospace* indicate placeholder names for variable information that must be supplied (i.e., arguments).

Note: *Due to management and file size considerations, only the first occurrence of the reference on each page is an active link. Subsequent references on the same page will not be actively linked to the definition and will use the standard, nonunderlined* **BOLD Monospace** *typeface. Find the first instance of the name (in the underlined* **BOLD Monospace** *typeface) on the page and click on the word to jump to the function or type definition.*

The following typographic conventions are used in this document to illustrate the Extended Backus-Naur Form.

[item]	Square brackets denote the enclosed item is optional.
{item}	Curly braces denote a choice or selection item, only one of which may occur on a given line.
<item>	Angle brackets denote a name for an item.
(range-range)	Parenthesis with characters and dash characters denote ranges of values, for example, (a-zA-Z0-9) indicates a single alphanumeric character, while (0-9) indicates a single digit.
"item"	Characters within quotation marks are the exact content of an item, as they must appear in the output text file.
?	The question mark denotes zero or one occurrences of an item.
*	The star character denotes zero or more occurrences of an item.
+	The plus character denotes one or more occurrences of an item.
item$^{\{n\}}$	A superscript number, n, is the number occurrences of the item that must be used. Example: (0-9)^{8} indicates that there must be exactly eight digits, so 01234567 is valid, while 1234567 is not valid.
item$^{\{n,\}}$	A superscript number, n, within curly braces followed by a comma "," indicates the minimum number of occurrences of the item, with no maximum number of occurrences.
item$^{\{,n\}}$	A superscript number, n, within curly braches, preceded by a comma ",”indicates a maximum number of occurrences of the item.
item$^{\{n,m\}}$	A super script number, n, followed by a comma "," and a number, m, indicates that the number of occurrences can be from n to m occurrences of the item, inclusive.

2
Design Discussion

This section of the document provides an overview to the build process for UEFI and PI compliant modules. This includes existing EDK components and new EDK II modules.

Note: *Path and Filename elements within the EDK II Meta-Data files and command line arguments are case-sensitive in order to support building on UNIX style operating systems.*

2.1 Reference Implementation: UEFI & PI formats

EDK II Build Tools follow both the UEFI and PI specifications for file formats. In some cases, the tools have been extended to follow the *Intel Innovation Framework for*

2.2 Development Environments

The EDK II build environment must support development workstations running Microsoft* Windows operating systems, or Linux operating systems or Apple Mac OS/X operating systems. In addition, multiple compiler tools chains from Microsoft, Intel and GCC must be supported. All provided source code should be Posix compliant. If assembly source code is used, both GCC (GAS) and Microsoft (MASM) style files should be provided. See the *EDK II C Coding Standard* for additional information.

2.2.1 Reference Implementation: Build process restrictions

The build process for all configurations must be identical, with the caveat that only applicable EDK II Packages need compile for any given operating system. An additional restriction on the workstation is the architecture of the workstation - only IA32 and X64 based workstations are supported. The EDK II BaseTools package, which provides the reference build system, can be built with binaries for Win32, Win64 (X64, not IPF), Linux32, Linux64 (only X64), Mac32 and Mac64 (X64). Some native versions of X64 operating systems may not be available.

2.3 UEFI/PI Firmware Images

UEFI and PI specifications define the standardized format for EFI firmware storage devices (FLASH or other non-volatile storage) which are abstracted into "Firmware Volumes." A Firmware Volume (FV) is a file level interface to firmware storage. Multiple FVs may be present in a single FLASH device, or a single FV may span multiple FLASH devices. An FV may be produced to support some other type of storage entirely, such as a disk partition or network device. For more information consult the *Platform Initialization Specification*, Volume 3.

In all cases, an FV is formatted with a binary file system. The file system used is typically the Firmware File System (FFS), but other file systems may be possible in some cases. Hence, all modules are stored as "files" in the FV. Some modules may be "execute in place" (linked at a fixed address and executed from the ROM), while others

are relocated when they are loaded into memory and some modules may be able to run from ROM if memory is not present (at the time of the module load) or run from memory if it is available.

Files themselves have an internally defined binary format. This format allows for implementation of security, compression, signing, etc. Within this format, there are one or more "leaf" images. A leaf image could be, for example, a PE32 image for a DXE driver.

Therefore, there are several layers of organization to a full UEFI 2.3/PI 1.2 firmware image. These layers are illustrated below in Figure 1. Each transition between layers implies a processing step that transforms or combines previously processed files into the next higher level. Also shown in Figure 1 are the reference implementation tools that process the files to move them between the different layers.

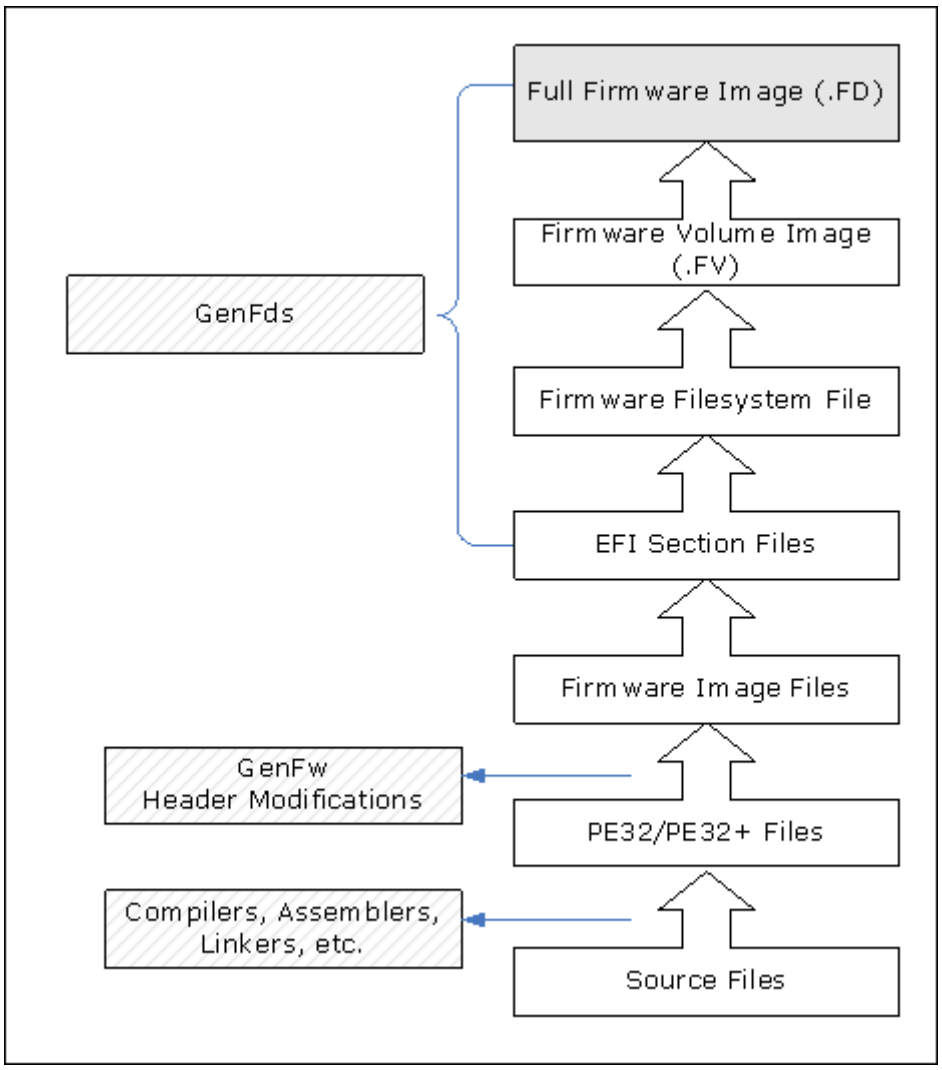


Figure 1. UEFI/PI Firmware Image Creation

In addition to creating images that initialize a complete platform, the build process also

supports creation of stand-alone UEFI applications (including OS Loaders) and Option ROM images containing driver code. Figure 2, below, shows the reference implementation tools and creation processes for both of these image types.

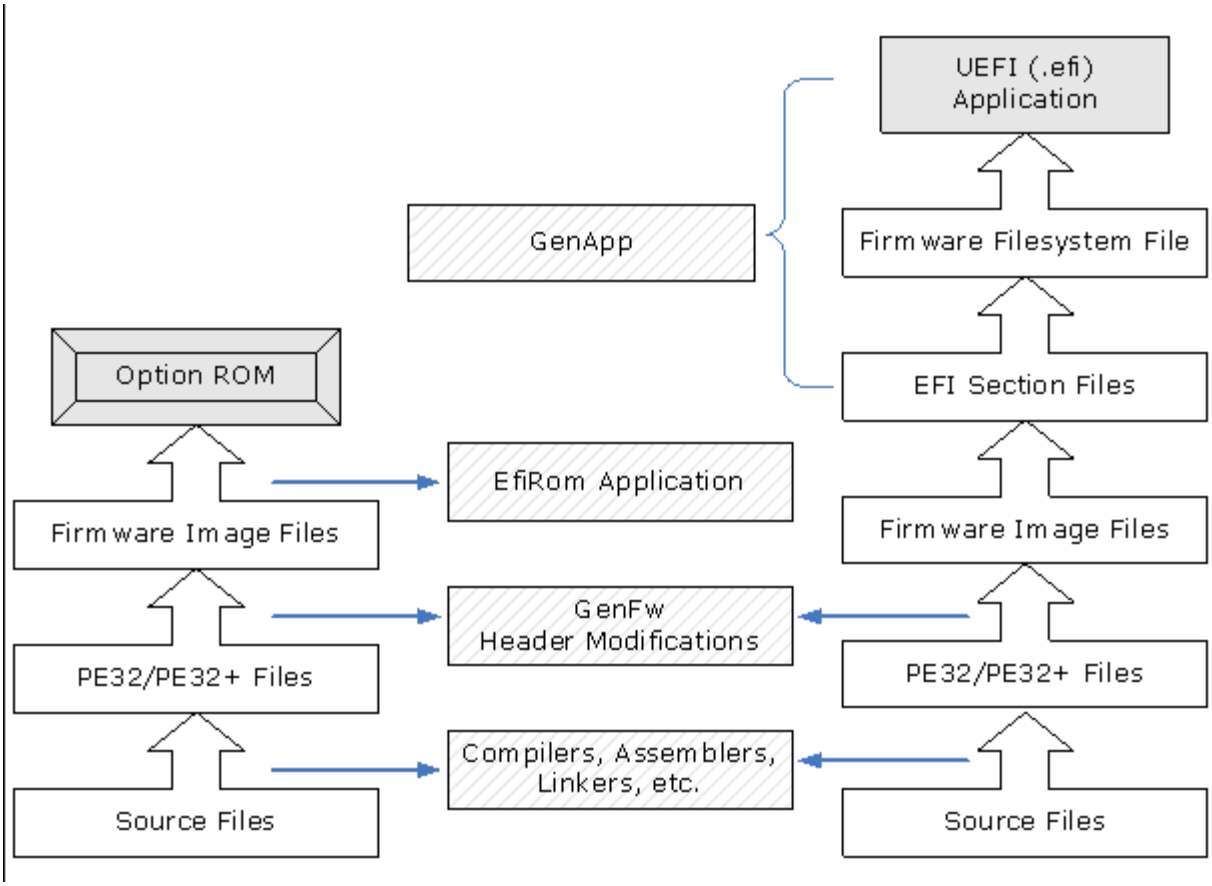


Figure 2. EFI PCI Expansion OptionROM and UEFI Application Creation

The final feature that is supported by the EDK II build process is the creation of Binary Modules that can be packaged and distributed for use by other organizations. Binary modules do not require distribution of the source code. This will permit vendors to distribute UEFI images without having to release proprietary source code.

This packaging process permits creation of an archive file containing one or more binary files that are either Firmware Image files or higher (EFI Section files, Firmware Filesystem files, etc.). The build process will permit inserting these binary files into the appropriate level in the build stages.

2.4 Boot Sequence

PI compliant system firmware must support the six phases: security (SEC), pre-efi initialization (PEI), driver execution environment (DXE), boot device selection (BDS), run time (RT) services and After Life (transition from the OS back to the firmware) of system. Refer to Figure 3 below.

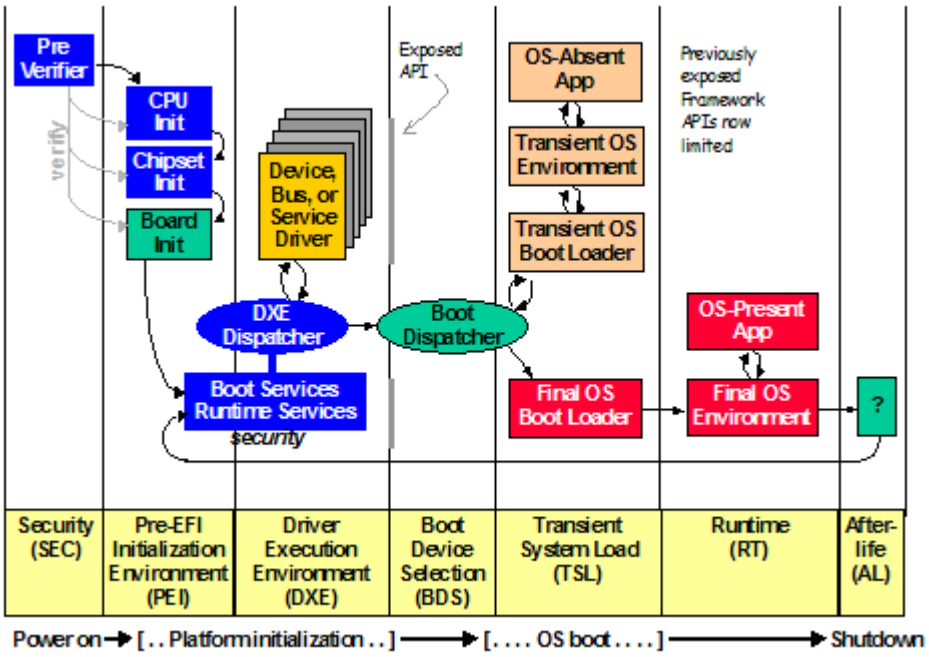


Figure 3. PI Firmware Phases

2.4.1 Security (SEC)

The Security (SEC) phase is the first phase in the PI Architecture and is responsible for the following:

- Handling all platform restart events
- Creating a temporary memory store
- Serving as the root of trust in the system
- Passing handoff information to the PEI Foundation

The security section may contain modules with code written in assembly. Therefore, some EDK II module development environment (MDE) modules may contain assembly code. Where this occurs, both Windows and GCC versions of assembly code are provided in different files.

2.4.2 Pre-EFI Initialization (PEI)

The Pre-EFI Initialization (PEI) phase described in the PI Architecture specifications is invoked quite early in the boot flow. Specifically, after some preliminary processing in the Security (SEC) phase, any machine restart event will invoke the PEI phase.

The PEI phase initially operates with the platform in a nascent state, leveraging only on-processor resources, such as the processor cache as a call stack, to dispatch Pre-EFI Initialization Modules (PEIMs). These PEIMs are responsible for the following:

- Initializing some permanent memory complement
- Describing the memory in Hand-Off Blocks (HOBs)
- Describing the firmware volume locations in HOBs

- Passing control into the Driver Execution Environment (DXE) phase

2.4.3 Drive Execution Environment (DXE)

Prior to the DXE phase, the Pre-EFI Initialization (PEI) phase is responsible for initializing permanent memory in the platform so that the DXE phase can be loaded and executed. The state of the system at the end of the PEI phase is passed to the DXE phase through a list of position independent data structures called Hand-Off Blocks (HOBs). HOBs are described in detail in the *Platform Initialization Specification*.

There are several components in the DXE phase:

- DXE Foundation
- DXE Dispatcher
- A set of DXE Drivers

2.4.4 Boot Device Selection (BDS)

The Boot Device Selection (BDS) phase is implemented as part of the BDS Architectural Protocol. The DXE Foundation will hand control to the BDS Architectural Protocol after all of the DXE drivers whose dependencies have been satisfied have been loaded and executed by the DXE Dispatcher. The BDS phase is responsible for the following:

- Initializing console devices
- Loading device drivers
- Attempting to load and execute boot selections

2.4.5 Transient System Load (TLS) and Runtime (RT)

The Transient System Load (TLS) primarily the OS vendor provided boot loader, along with the Runtime Services (RT) phases may access persistent UEFI drivers and applications. Drivers in this category include PCI Option ROMs.

2.4.6 After Life (AL)

The After Life (AL) phase consists of persistent UEFI drivers used for storing the state of the system during the OS orderly shutdown, sleep, hibernate or restart processes.

2.5 Typical Flash Part Layout

While a flash part layout is specific to a given platform, there are some generalizations that apply. The SEC and PEI code is typically put into a "RECOVERY" location, while all remaining sections are put into the "MAIN" location. The MAIN section may be compressed for size optimization, provided the PEI or SEC code contains appropriate decompression drivers. The PI specification defines only standard EFI compression; if other compression mechanisms (or verification mechanisms, such as CRC32) are required, then both the tools for creating a compressed image and a library for decompressing the image must be provided. These non-standard compression, encryption, signing or verification mechanisms are applied to a GUIED encapsulation section. Each method needs a unique GUID, however the methods may be applied to images more than once per FD image. This is done in order to facility recovery and updates (called capsules). Other areas in flash may be reserved for non-volatile (NV)

data storage, fault tolerant working (FTW) space or vital product data (VPD) areas. These other regions are not defined in the PI specification, and implementation is left to the platform integrator. The reference design Nt32 Platform emulation environment contains a virtual flash device. The content within this virtual FD is laid out per Figure 4.

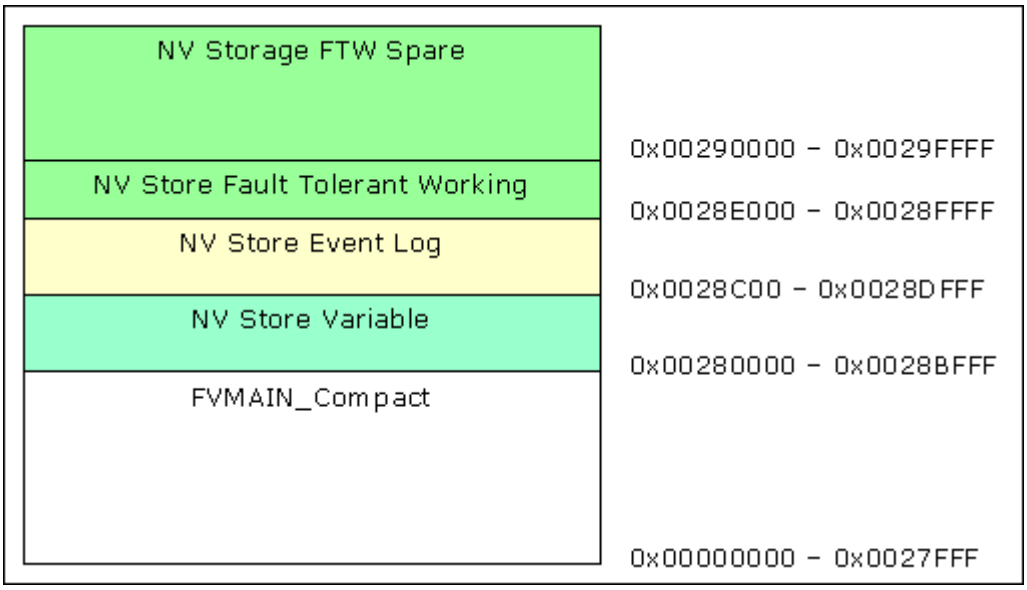


Figure 4. NT32 Flash Device Layout

Figure 5 represents a typical IA32/X64 FD layout, where SEC and PEI code is located in the FV Recovery section, and the remaining drivers are located in a GUIDED encapsulation (compressed) section designated as FVMAIN_Compact.

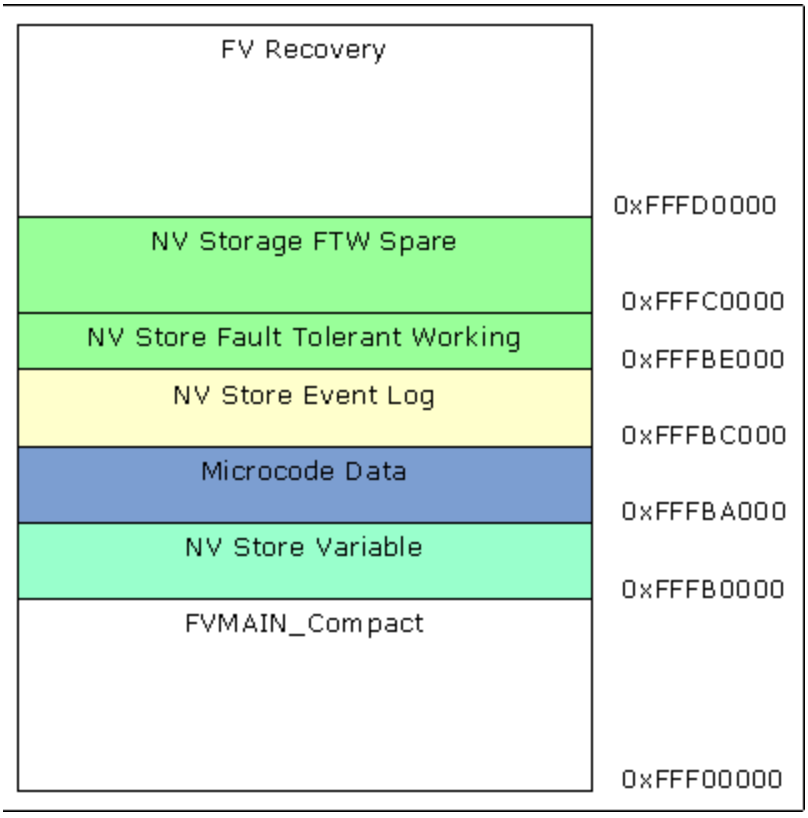


Figure 5. Typical IA32/X64 Flash Device Layout

Figure 6 represents a typical IPF FD layout.

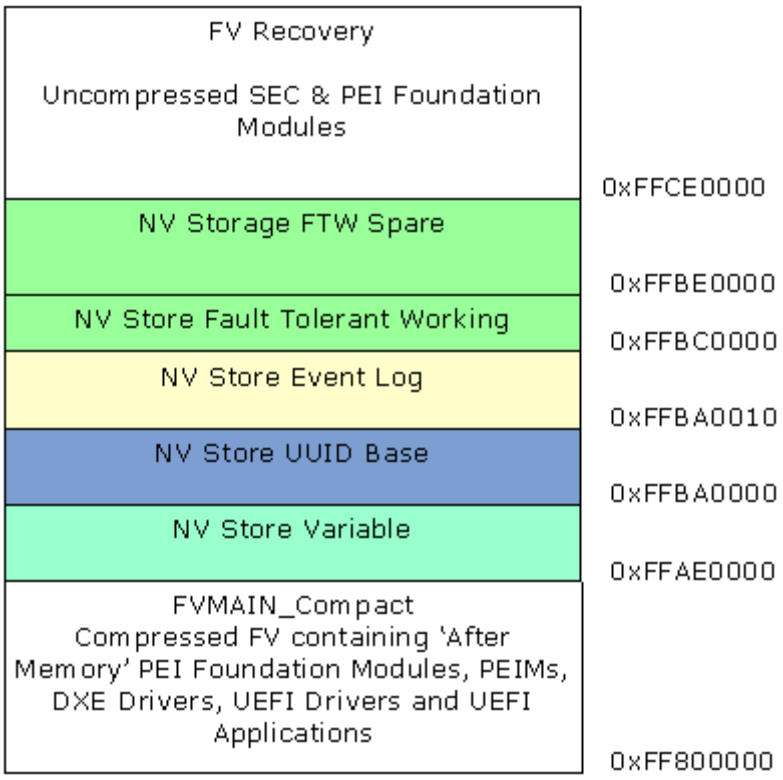


Figure 6. Typical IPF FD Layout

All of these layouts assume only one flash device, with the virtual memory addresses listed for each section within the FD.

Note: *More than one flash device may be present within a platform, so the images may be split over multiple devices.*

2.6 Generic Build Process

All code starts out as either C sources and header files, assembly sources and header files or binary data (native instructions, such as microcode) files. Per the UEFI and PI specifications, the C and Assembly files must be compiled and linked into PE32/PE32+ images.

While some code is designed to execute only from ROM, most UEFI/PI modules are written to be relocate-able. These are written and built different. For example, Execute In Place (XIP) module code is written and compiled to run from ROM, while the majority of the code is written and compiled to execute from memory, which requires that the code be relocate able.

Some modules may also permit dual mode, where it will execute from memory only if memory is available, otherwise it will execute from ROM. Additionally, modules may permit dual access, such as a driver that contains both PEI and DXE implementation code. Code is assembled or compiled, then linked into PE32/PE32+ images, the relocation section may or may not be stripped and an appropriate header will replace

the PE32/PE32+ header. Additional processing may remove more non-essential information, generating a Terse (TE) image.

The binary executables are converted into EFI firmware file sections. Each module is converted into an EFI Section consisting of an Section header followed by the section data (driver binary).

2.6.1 EFI SECTION Files

The general section format for sections less than 16MB in size is shown in Figure 7. Figure 8 shows the section format for sections 16MB or larger in size using the extended length field.

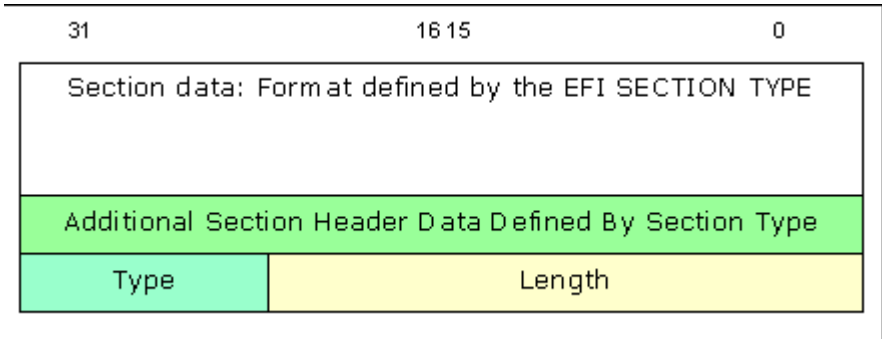


Figure 7. General EFI Section Format (< 16MB)

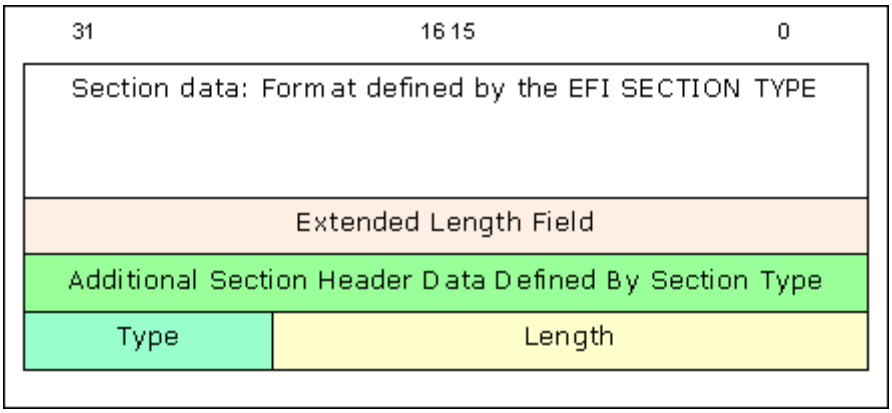


Figure 8. General EFI Section Format for Large Size Sections.

Table 2 below lists the different architecturally defined section types, refer to the *PI Specification*, Volume 3 for additional details.

Table 2. EFI Section Types

Name	Description
EFI_SECTION_COMPRESSION	Encapsulation section where other sections are compressed
EFI_SECTION_GUID_DEFINED	Encapsulation section where other sections have a format defined by a GUID.
EFI_SECTION_DISPOSABLE	Encapsulation section used during the build process but not required for execution.
EFI_SECTION_PE32	PE32+ Executable Image
EFI_SECTION_PIC	Position-Independent Code.
EFI_SECTION_TE	Terse Executable image.
EFI_SECTION_DXE_DEPEX	DXE Dependency Expression.
EFI_SECTION_VERSION	Version, Text and Numeric (UNICODE)
EFI_SECTION_SMM_DEPEX	Leaf section type for determining the dispatch order for an SMM driver.
EFI_SECTION_USER_INTERFACE	User-Friendly name of the driver (UNICODE)
EFI_SECTION_COMPATIBILITY16	DOS-style 16-bit executable.
EFI_SECTION_FIRMWARE_VOLUME_IMAGE	PI Firmware Volume Image.
EFI_SECTION_FREEFORM_SUBTYPE_GUID	Raw data with GUID in header to define format.
EFI_SECTION_RAW	Raw data (for example, a logo).
EFI_SECTION_PEI_DEPEX	PEI Dependency Expression.

2.6.2 Firmware Files

Multiple EFI Sections are combined into a Firmware file (FFS) which consists of zero or more EFI sections. Each FFS consists of a FFS header plus the data. Figure 9 show the basic FFS File layout and Figure 10 shows the FFS File layout for files of 16MB or larger.

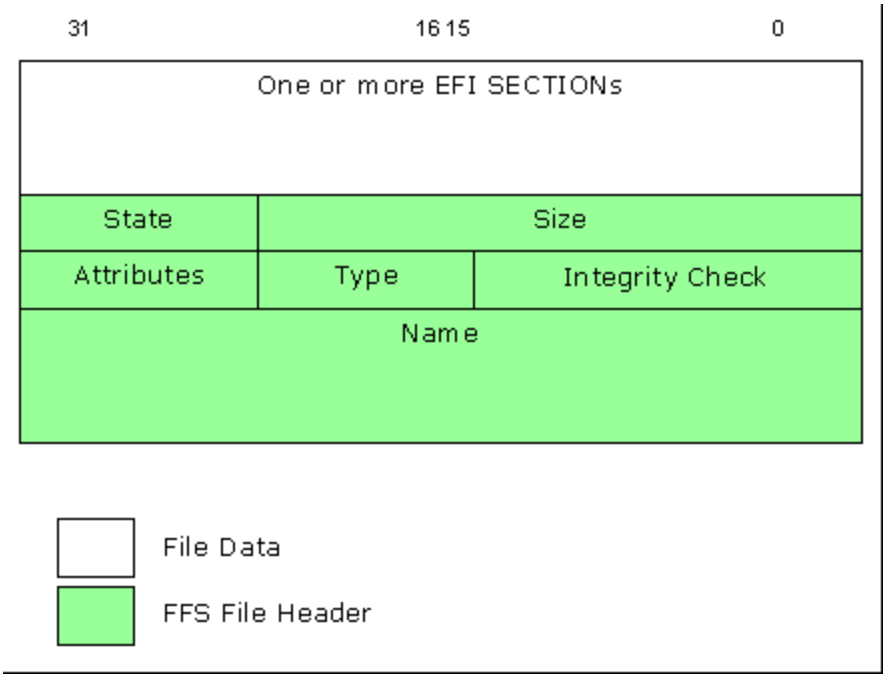


Figure 9. Typical FFS File Layout (<16MB)

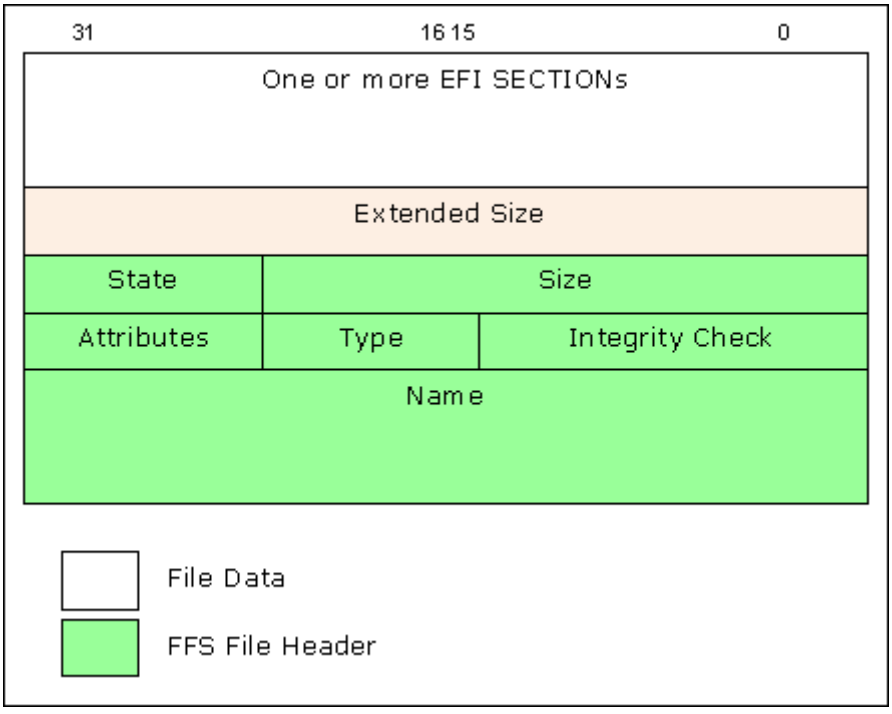


Figure 10. File Header 2 layout for files larger than 16Mb

Table 3 lists the different FV file types architecturally defined in the PI specification

describing the content (FFS) of the Firmware Volume Data.

Table 3. Defined FV File Types

Name	Description	Code
EFI_FV_FILETYPE_RAW	Binary data.	0x01
EFI_FV_FILETYPE_FREEFORM	Sectioned Data.	0x02
EFI_FV_FILETYPE_SECURITY_CORE	Platform core code used during the SEC phase.	0x03
EFI_FV_FILETYPE_PEI_CORE	PEI Foundation code.	0x04
EFI_FV_FILETYPE_DXE_CORE	DXE Foundation code.	0x05
EFI_FV_FILETYPE_PEIM	PEI Module (PEIM)	0x06
EFI_FV_FILETYPE_DRIVER	DXE driver.	0x07
EFI_FV_FILETYPE_COMBINED_PEIM_DRIVER	Combined PEIM/DXE driver	0x08
EFI_FV_FILETYPE_APPLICATION	Application	0x09
EFI_FV_FILETYPE_SMM	Contains a PE32+ image that will be loaded into SMRAM.	0x0A
EFI_FV_FILETYPE_FIRMWARE_VOLUME_IMAGE	An embedded Firmware Volume Image.	0x0B
EFI_FV_FILETYPE_COMBINED_SMM_DXE	Contains PE32+ image that will be dispatched by the DXE Dispatcher and will also be loaded into SMRAM.	0x0C
EFI_FV_FILETYPE_SMM_CORE	SMM Foundation	0x0D
EFI_FV_FILETYPE_OEM_*	OEM File Types	0xC0 – 0xDF
EFI_FV_FILETYPE_DEBUG_*	Debug/Test File Types	0xE0 – 0xEF
EFI_FV_FILETYPE_FFS_*	Firmware File System Specific File Types	0xF0 – 0xFF
EFI_FV_FILETYPE_PAD	Pad file for FFS.	0xF0

2.6.3 Firmware Volumes

One or more FFS files are combined into a Firmware Volume (FV). The format for a an FV is a header followed by an optional extended header, followed by zero or more FFS files. Figure 11 illustrates the layout of the FV.

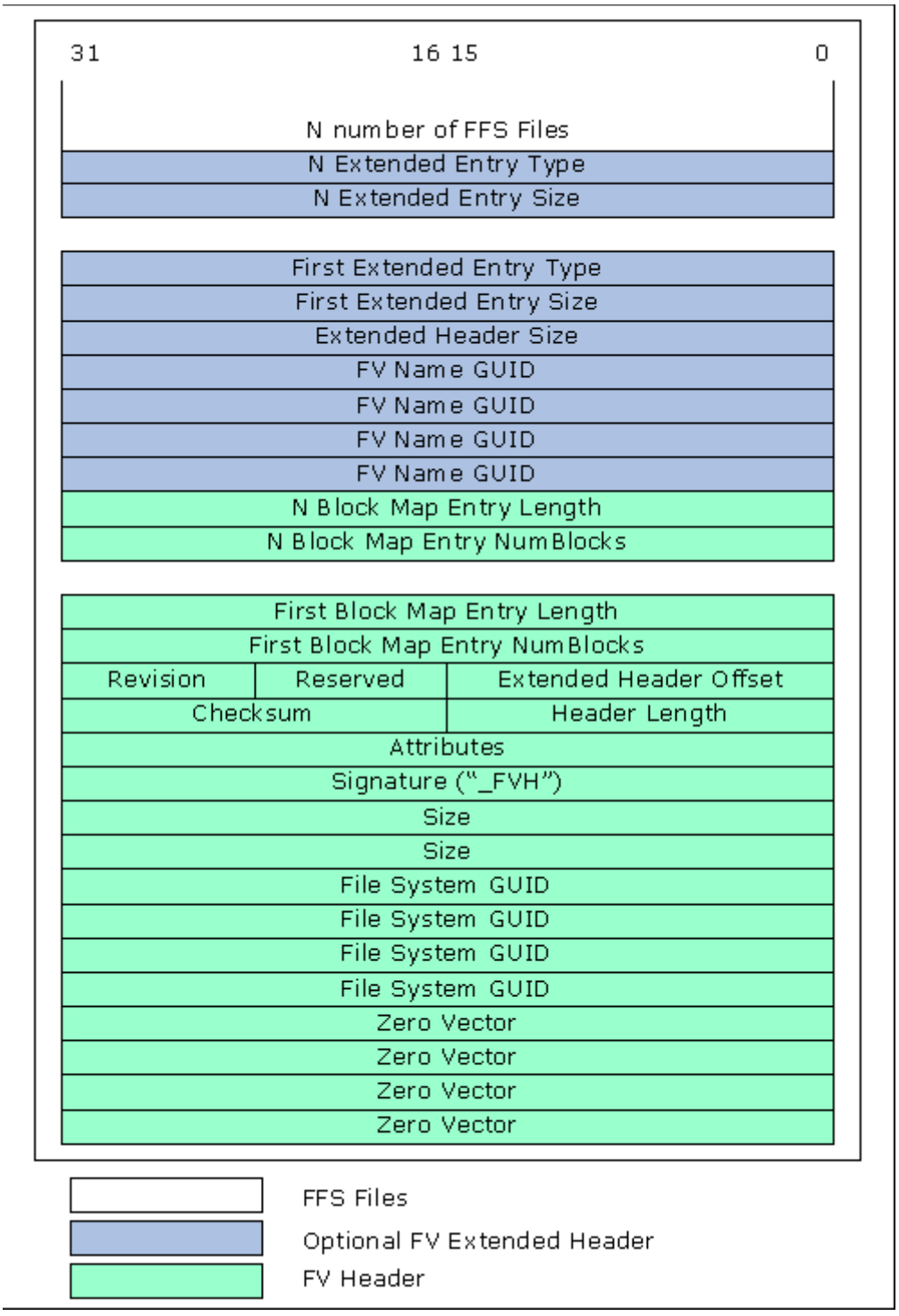


Figure 11. General FV Layout

Multiple FV files, each of which is just a logical firmware device, can be combined into a single FD image.

Within the context of modules, error messages within the code are written in plain text (English - ASCII) while messages that are displayed as part of the menu system or are stored for display later, are written in UNICODE (UCS-2 encoded) format. The UEFI/PI specifications define the structure for Human Interface Infrastructure (HII) as well as Visual Forms Representation (VFR). Also supported, but not defined, within the reference build and the UEFI/PI specifications is the user defined, Vital Product Data (VPD) areas.

2.6.4 Special Files - VTF & BSF

The Volume Top File (VTF) is a file that must be located such that the last byte of the file is also the last byte of the firmware volume. Regardless of the actual file type, a VTF file must have the file name GUID of EFI_FFS_VOLUME_TOP_FILE_GUID. The file name is a GUID, andEFI_FFS_VOLUME_TOP_FILE_GUID is the C define that is used by code and the build system in place of the GUID value.

The build system must be aware of this GUID and insert a pad file if necessary to guarantee the VTF is located correctly at the top of the firmware volume. This is also required for update and write operations.

The Bootstrap file is firmware file that is aligned to the top of the 32-bit address space. It is responsible for encapsulating the reset vector for the Itanium processor family and IA-32. It also contains fixed information, such as the PEIM return link for IA-32 and the entry point to the PEI core. Also of interest, it contains the base of the boot FV to enable successive module discovery in PEI.

2.6.5 EFI_FV_FILETYPE_SECURITY Notes

The security section is always executed from ROM. For size optimization, the relocation (.reloc) section of security executables may be stripped. The reference build does this by default.

Security drivers run directly from flash need to have the BaseAddress re-based to the location the driver occupies in ROM prior to putting the driver into a Firmware Volume (FV).

2.6.6 EFI_FV_FILETYPE_PEI_CORE Notes

The last step of the security section was to hand-off execution to the PEI foundation, which is typically executed in three phases, pre-memory, during memory detection and after memory is available. For size optimization, it is recommended to have the pre-memory and memory detection PEI core modules ROM resident, to have the PE32+ image converted to a terse image, and to have the .reloc section stripped. After memory is present, the PEI Core modules should be shadowed in memory to speed up execution. These modules can also contain signing, decryption and/or decompression routines to handle verification, uncompressing or decrypting algorithms for GUIDED encapsulation sections or for compressed PEIMs and any remaining FVs that contain the DXE Foundation and all drivers and applications that are used in the DXE phase or later. The decompression should always occur after memory is available.

The PEI Foundation modules that run directly from flash, need to have the BaseAddress re-based to the location it occupies in ROM, prior to putting the driver into an FV. By default, the reference build system will strip the .reloc section of all modules.

2.6.7 EFI_FV_FILETYPE_PEIM Notes

There are three types of PEIMs:

- XIP must execute from ROM,
- PEIMs that must be executed from memory and
- PEIMs that will execute in from memory if memory is available. If no memory is available, then the PEIMs can execute from ROM.

Some PEIMs are executed from ROM, some PEIMs are executed only from memory and some PEIMs may be executed from memory only if memory is present.

For PEIMs executed only from ROM, the image should be converted to a terse image, the .reloc section stripped for size optimization and module cannot be compressed - the images must be re-based to the location in ROM.

PEIMs that execute from memory should never have the .reloc section stripped, but may be converted to terse images and may be compressed.

PEIMs that are coded to register for shadow, i.e., they may be run from memory if memory is present, should not have the .reloc section stripped. The reference build implementation uses a keyword, SHADOW, in the module's INF file to indicate this mode, setting **SHADOW = TRUE**. By default, the reference build implementation will strip the .reloc section of PEIMs; PEIMs must specify the **SHADOW = TRUE** in the module's INF file to prevent this. Additional flags in the FDF file, **RELOCS_RETAINED** and **RELOCS_STRIPPED**, are provided to over-ride stripping of the .reloc section.

Like the PEI Foundation, PEIMs that can, should be shadowed in memory to speed up execution.

Once the PEI Foundation has been loaded, PEIMs are dispatched, and if a PEIM is dependent on the existence of another PEIM, an **EFI_SECTION_PEI_DEPEX** section is used to define the dependency relationship. The PEI Foundation will use this section (if present in an FFS) to ensure the required PEIMs are available prior to dispatch.

2.6.8 EFI_FV_FILETYPE_COMBINED_PEIM_DRIVER Notes

Dual function (PEI/DXE) drivers (PEIMs that are coded to register for shadow) should never have the .reloc section stripped. Additionally, compression of these modules may decrease the overall size of the FD image in hardware. Using the terse image format for drivers of this type is not permitted by the PI specification. For this class of driver, one PEI and/or one DXE dependency section can be added to the FFS file containing the image.

2.6.9 DXE, BDS, TLS and AL Notes

Stripping the .reloc section from these modules and any UEFI applications is not recommended, but is allowed in certain cases. Additionally, these images cannot be converted to the terse format - only elements of the PEI Foundation (PEI Core) and PEIMs can be converted to use the terse format headers. Compression of the images is permitted, however, as most compression algorithms work better over a larger data set, it is recommended that the images be combined into a Firmware Volume, and the entire FV can be compressed.

The modules (after the DXE Foundation has been given control) may have other dependent drivers. Similar to the **EFI_SECTION_PEI_DEPEX** section, a dependency **EFI_SECTION_DXE_DEPEX** section may be required. These files are used by the DXE

foundation to ensure required drivers are available when needed.

Another feature of some of these modules, the BDS is particular, has to do with the Human Interface Infrastructure (HII). The HII uses internal forms representation (IFR) coded files.

2.7 Creating EFI Images

2.7.1 Compiling Code

EDK II modules include both libraries, drivers and applications. Library modules are compiled and linked as static libraries. Drivers and applications are compiled to object files, then linked with the static libraries they require. After the static image has been created, the resulting image is run through the dynamic linker to generate the relocate-able binary images (DLL). All EFI images must be formatted PE32/PE32+/COFF.

Note: *ELF images created by GCC on Linux systems need additional processing to simulate the PE32+/ COFF format.*

Since UEFI/PI images are not standard executables, these dynamically linked (DLL) files must be processed to become UEFI/PI compliant images. This processing involves replacing the standard header with an EFI header that reflects the EFI_SECTION type. Prior to creating the EFI section files, PEI Foundation and PEIM images may be processed into either a terse image, or have the .reloc section removed (for images that will always execute directly from ROM).

2.7.1.1 Reference Implementation: "Best Known Safe" flags

The reference build provided with EDK II provides files for the supported (tested) compiler tool chains that contain "Best Known Safe" flags for generating libraries, drivers and applications. These flags are set for speed optimization. The build system does provide for modifying flags for individual modules, so flags may be modified for better debugability.

2.7.2 Creating a Terse Image

The following is an partial view of the process; ommissions may exist.

To create a Terse image:

1. The DOS, PE and/or optional headers must be replaced with a minimal header. The TE header will have a signature of "VZ". Per the PE/COFF specification, at offset 0x3C in the file is a 32-bit offset (from the start of the file) to the PE signature, which always follows the MSDOS stub. The PE signature is immediately followed by the COFF file header.
2. After verifying the DOS header's magic number (0x5A4D), the PE signature ("PE\0\0")is verified, then obtains the Machine type from the optional header's subsystem field.
   - Since this process removes bytes from the file, the number of bytes stripped must be calculated based on the location of the PESigOffset (0x3C) plus the 4 bytes containing the offset pointer plus the size of the Coff header plus the size of any Optional Header (from the Coff header's SizeOfOptionalHeader).
   - The number of bytes stripped must always be less than 64K bytes. The original file size minus the number of bytes stripped is then inserted into the TE header's StrippedSize value.

- The optional header's magic number is used to determine whether the Optional Header data structure is IPF (**EFI_OPTIONAL_IMAGE_HEADER64**) or non-IPF (**EFI_OPTIONAL_IMAGE_HEADER32**) .
- Using the correct optional header format, the TE header's **AddressOfEntryPoint** is set to the optional header's **AddressOfEntryPoint**. Additionally, the **Subsystem** entry from the optional header's **Subsystem** entry will be packed into one byte.
- Additional entries, **BaseOfCode** and **ImageBase** in the TE header come from the optional Header. If the optional header's **NumberOfRvaAndSizes** is greater than 0, then the relocation data from the optional header **DataDirectory[0].VirtualAddress** and **Size** is set based on the content of the optional header's **DataDirectory[0]** values.
- Likewise, if the **NumberOfRvaAndSizes** is greater than 1, then the debug data from the optional header's **DataDirectory[1].VirtualAddress** and **Size** is set in the TE header's **DataDirectory[1]** entry.

3. As a last step before creating the image, the COFF header specifies the value of the **NumberOfSections** in the file which needs to be packed into a single byte of the TE header. The number of sections must be less than 255 for this to succeed.
4. After the header is created, then the rest of the original image - all header information stripped is appended to the TE header.

Figure 10 shows the relationship of original image to the TE image.

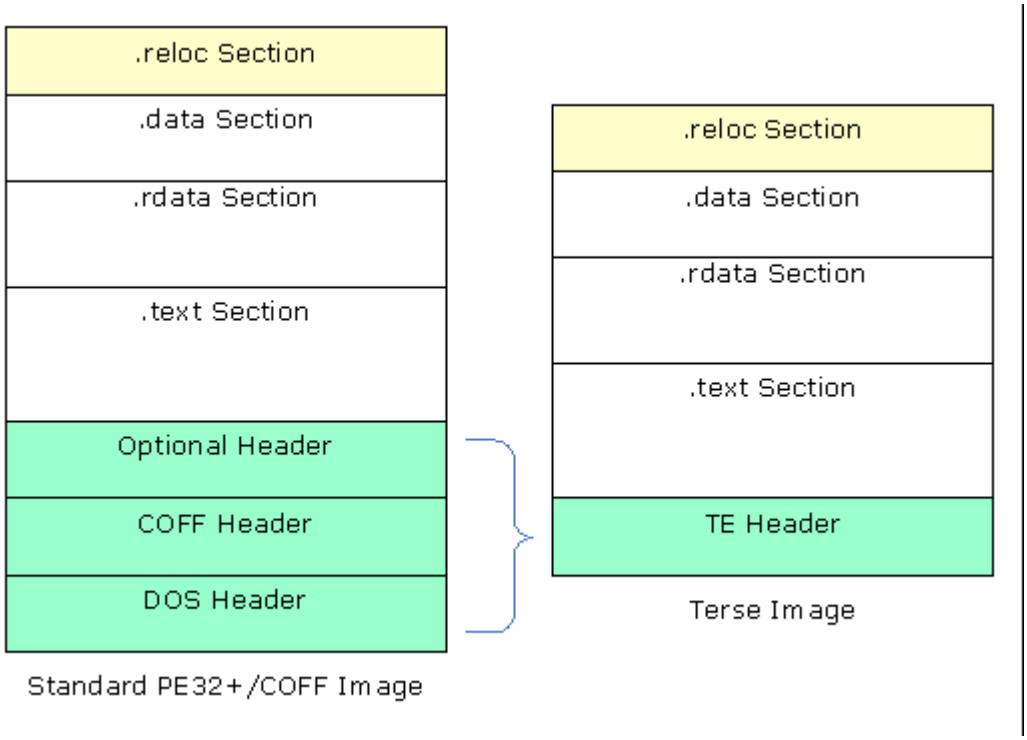


Figure 12. Standard Image to Terse Image Comparison

2.7.3 Removing reloc sections

Removing the relocation section of either a PE or TE image can only be done if the .reloc section is at the end of the file. While most .reloc sections are fairly small in comparison to the other sections of the files, removing all of the .reloc sections in combination with using Terse images for the PEI foundation and PEIMs that do not register for shadow (see UEFI/PI specs) can shrink a platform image by several hundred bytes.

For a TE image, the file size (StrippedSize) is adjusted by subtracting the length of the .reloc section. The DataDirectory[0] VirtualAddress is set to 0, as is the Size parameter.

Removing the relocation section of a PE image is slightly more complicated. The PE32+ image header (which contains both the EFI_IMAGE_DOS_SIGNATURE - 0x5A4D and an EFI_IMAGE_NT_SIGNATURE - 0x00004545) will be modified by setting the EFI_IMAGE_FILE_HEADER Characteristics' EFI_IMAGE_FILE_RELOCS_STRIPPED bit. The IPF header uses a similar data structure to IA32, X64 and EBC data structures. The naming within the data structures is consistent. Therefore, regardless of the machine type, both the SizeOfImage and SizeOfInitializedData are adjusted by subtracting the length of the .reloc section. If the NumberOfRvaAndSizes is greater than the EFI_IMAGE_DIRECTORY_ENTRY_BASERELOC, then the DataDirectory[0] VirtualAddress and the Size are both set to 0. Finally, the .reloc section's header is modified, setting the Misc.VirtualSize and SizeOfRawData to 0.

2.7.4 Generating LEAF EFI_SECTION Files

This section provides the overview for generating EFI SECTION files. EFI_SECTION headers must be present on all leaf sections. The EFI Section header (see above) will be prefixed to the file or data section (VERSION and USER_INTERFACE data can be generated "on-the-fly" rather than creating a separate UNICODE file first).

For the files that are PE32 code (EFI_SECTION_PE32, EFI_SECTION_TE and EFI_SECTION_PIC, the .text, .debug, .reloc and .data section headers (if they exist) are overwritten with the EFI_IMAGE_SECTION_HEADER.

The section name (i.e., .text) is copied into the Name entry, while the remaining sections are set as follows:

```
Hdr->Misc.VirtualSize = Size;
Hdr->VirtualAddress = Offset;
Hdr->SizeOfRawData = Size;
Hdr->PointerToRawData = Offset;
Hdr->PointerToRelocations = 0;
Hdr->PointerToLinenumbers = 0;
Hdr->NumberOfRelocations = 0;
Hdr->NumberOfLinenumbers = 0;
Hdr->Characteristics = Flags;
```

- For a .text section, the Flags value is a bit-wise OR of EFI_IMAGE_SCN_CNT_CODE, EFI_IMAGE_SCN_MEM_EXECUTE and EFI_IMGE_SCN_MEM_READ (0x60000020).
- For a .data section, the Flags value is a bit-wise OR of EFI_IMAGE_SCN_CNT_INITIALIZED_DATA, EFI_IMAGE_SCN_MEM_WRITE and EFI_IMAGE_SCN_MEM_READ (0xC0000040).
- For a .reloc section, the Flags value is a bit-wise OR of EFI_IMAGE_SCN_CNT_INITIALIZED_DATA, EFI_IMAGE_SCN_MEM_DISCARDABLE and EFI_IMAGE_SCN_MEM_READ (0x42000040).
- For a .debug section, the Flags value is a bit-wise OR of EFI_IMAGE_SCN_CNT_INITIALIZED_DATA, EFI_IMAGE_SCN_MEM_DISCARDABLE and EFI_IMAGE_SCN_MEM_READ (0x42000040).

Once these have been modified, the EFI_COMMON_SECTION_HEADER will be prefixed to the file.

Each EFI_COMMON_SECTION_HEADER "type" field defines the data that follows. Table 4 lists

the section type value. All EFI section files start with the **EFI_COMMON_SECTION_HEADER**.

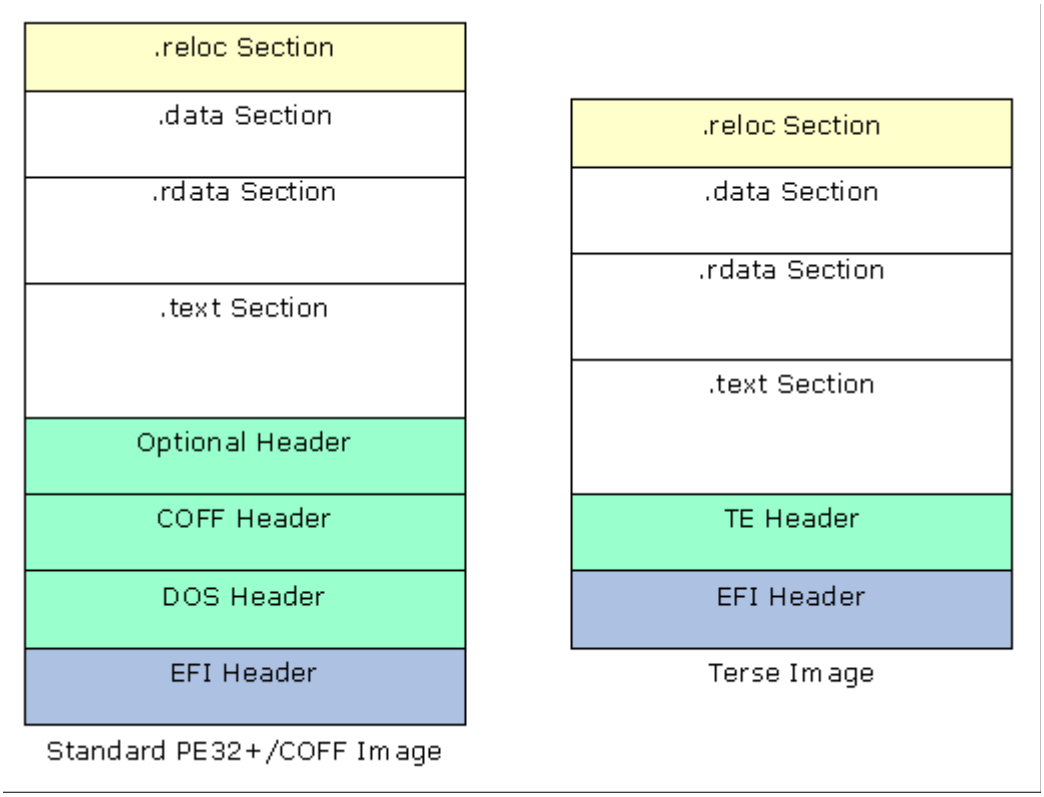


Figure 13. EFI Image Files

Table 4. Basic EFI_SECTION Type Codes

Section Type	Value
EFI_SECTION_PE32	0x10
EFI_SECTION_PIC	0x11
EFI_SECTION_TE	0x12
EFI_SECTION_VERSION	0x14
EFI_SECTION_USER_INTERFACE	0x15
EFI_SECTION_COMPATIBILITY16	0x16
EFI_SECTION_FIRMWARE_VOLUME_IMAGE	0x17
EFI_SECTION_FREEFORM_SUBTYPE_GUID	0x18
EFI_SECTION_RAW	0x19

The size for these standard sections is defined as a 24-bit unsigned integer that contains the total size of the section in bytes, including the **EFI_COMMON_SECTION_HEADER**. For example, a zero-length section has a *Size* of 4 bytes.

Except for the **EFI_SECTION_VERSION** and the **EFI_SECTION_USER_INTERFACE**, the format of each section is the **EFI_COMMON_SECTION_HEADER** prefixed to a file containing data. Refer to the definitions in Section 3 for **EFI_SECTION_VERSION** and **EFI_SECTION_USER_INTERFACE**.

2.7.5 Generating Encapsulation EFI_SECTION Files

This section provides the overview for generating the two Encapsulation EFI SECTION files. The **EFI_SECTION** header must be present along with additional header information. The encapsulation EFI Section header will be prefixed to the file or data section. There are three encapsulation **EFI_SECTION** types. The first two types listed have extended header information.

Table 5. Encapsulation EFI_SECTION Type Codes

Section Type	Value
EFI_SECTION_COMPRESSION	0x01
EFI_SECTION_GUID_DEFINED	0x02
EFI_SECTION_DISPOSABLE	0x03

A compression section uses the **EFI_SECTION_COMPRESSION** header, while the GUID defined section uses an **EFI_SECTION_GUID_DEFINED** header.

The size for these sections is defined as a 24-bit unsigned integer that contains the total size of the section in bytes, including the size of the header.

For the **EFI_SECTION_COMPRESSION**, the **CompressionType** field must be set to 0x01 for standard compression, or 0x00 if the image is not compressed.

Note: *In the specification, only* **PI_STD** *compression is supported for this section type.*

For the **EFI_GUID_DEFINED_SECTION**, which is used for non-standard compression (see above) the named GUID that defines the section follows the **EFI_COMMON_SECTION_HEADER**. After this GUID are two additional **UINT16** parameters, the first is the **DataOffset** which contains the offset in bytes from the beginning of the common header to the first byte of the data. An *Attributes* parameter is a bit field code which declares specific characteristics of the section contents.

These headers are prefixed to the data files, which may include the standard PE32 headers.

2.7.6 Generating DEPEX EFI_SECTION Files

This section provides the overview for creating PEI, DXE and SMM DEPEX sections. The DEPEX grammar is defined in the *PI Specification* Volume 1, *Dependency Expression Grammar* chapter, while the Opcodes for the Mnemonic are defined in Volume 2, *DXE Dispatcher* chapter. The translation of the mnemonic and/or GUID involves creating a binary file using postfix notation. The file does not conform to PE32+/COFF, and no header information is attached prior to generating the EFI_SECTION files. The format of the binary data is 8-bit aligned, with a single byte per opcode, with op-codes that require a GUID value (BEFORE, AFTER and PUSH) being followed by 16 bytes to contain the GUID value. See Table 6 below.

Table 6. Dependency Section Type Codes

Section Type	Value
`EFI_SECTION_PEI_DEPEX`	0x1B
`EFI_SECTION_DXE_DEPEX`	0x13
`EFI_SECTION_SMM_DEPEX`	0x1C

Once the binary file is created an EFI_SECTION file can be created, and the `EFI_COMMON_SECTION_HEADER` will be prefixed to the file.

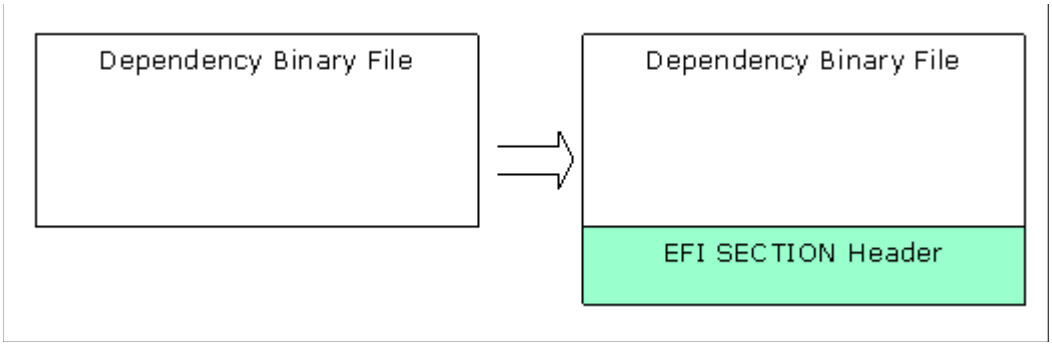


Figure 14. Depex File

2.7.6.1 Reference Implementation Guidelines

Use of a separate file for describing the dependencies is discouraged. Grammar of the INF, DSC and FDF files permit specifying the dependency expressions. Libraries may also have a dependency, `[depex]`, section. These dependencies should be appended to the module's DEPEX sections unless the module includes a depex (.dxs) file - even if the module does not contain a `[depex]` section. When a developer chooses to write the .dxs file, the developer is responsible for specifying all dependencies in the .dxs file.

Libraries that are linked to a UEFI DRIVER may have DEPEX sections. There are three 'rules' for the tools.

- Tools are coded so that for a given module the depex sections of all linked-in library instances are logically AND'd with the DEPEX section of the module
  - If no DEPEX section is specified in the module, then only the library instances DEPEX sections are logically AND'd to create the DEPEX section for the module
- Tools are also coded to ignore the depex sections of libraries that are linked to UEFI_DRIVER or PCI Option ROM code
- The tools should break the build if one module, using one of the noted module types, contains a depex section in the INF file.

2.7.7 Generating Visual Forms (IFR - HII) Files

This section covers the generation of the Human Interface Infrastructure (HII) format files used for displaying information on the console. While all error messages from the EFI drivers are written in English, displaying data on the console - selection and configuration menus - is performed using HII formats. This permits the user to select an alternate language for these displays.

Strings intended for these displays must be written in Unicode (UCS-2) format, rather

than plain ASCII text. The Unicode strings for these forms should be kept in separate files (.uni extension), or optionally, within C code (either .c source or .h header) files. Forms, Strings, fonts and images are stored in an HII database encoded to an Internal Forms Representation (IFR) - with each object and attribute a byte stream.

All HII files are included as part of a driver module's code - the data that makes up IFR content is compiled into standard object code and linked in to the driver.

2.7.7.1 Reference Implementation: Tools

The reference build system provides tools for processing formatted Unicode files and Visual Forms Representation (VFR) files in order to create the IFR files. Refer to the EDK II User's Manual for more information regarding the use of the Unicode and VFR files. Refer to the VfrCompiler description and VFR BNF later in this document for more detailed information on the provided implementation. Additionally, the reference build AutoGen tools are used to process Unicode files listed in a module's INF file. Also note that the IFR code is not compatible - UFI compliant IFR code is different from the IFR code defined by early Intel Framework documents.

2.7.7.2 Reference Implementation: More than One Unicode File

If more than one Unicode file is required by a module, the rules for including these files are as follows. If one Unicode file uses a #include statement to include other Unicode files, only the top-level Unicode file should be listed. These secondary Unicode files must not be listed in the INF file.

2.7.7.3 Reference Implementation: Compatability

The EDK II Vfr compiler tools can process EDK and EDK II VFR and Unicode files and to generate UEFI/PI compliant IFR files. EDK II Unicode files can use the UEFI defined Unicode extended grammar. The EDK VFR and Unicode files are a subset of the EDK II versions. EDK II VFR and Unicode files may not be used with an EDK build unless they do not include the extended grammar. Table 7 shows the compatibility matrix.

Table 7. VFR Compatibility Matrix

Code	non-UEFI 2.1 Compliant VFR Tools	UEFI 2.3 Compliant VFR Tools
pre-UEFI 2.1 Unicode	Yes	Yes
pre-UEFI 2.1 VFR Source	Yes	Yes
pre-UEFI 2.1 IFR – binaries	Yes	No
UEFI 2.1 Unicode	No	Yes
UEFI 2.1 Vfr	No	Yes
UEFI 2.1 IFR – binaries	No	Yes

2.7.8 Generating EFI FFS Files

This section provides the overview for generating an FFS file. Once the EFI Section files have been created, they need to be placed within an FFS file. An FFS file contains an FFS header and one or more section files. The ordering of the section files within the FFS is not specified by the PI specification, so sections may appear in any order. The Name of the FFS file, which is placed in the FfsFileHeader data structure is a GUID value with a structure of U`INT64`, `UINT32`, `UINT32`, `UINT8[8]`.

The alignment of data within the FFS must match the alignment specified for a given section, so padding may be required between the FFS header and the section headers. Alignment must be set and padding inserted prior to calculating the size or performing the Integrity check (checksum on the header itself and all of the section data).

The size of the FFS, in the `FfsFileHeader.Size` array is computed using the size of all files, including all pad files, plus the size of the header. The size value must be less than 0x01000000 (16MB).

The `FfsFileHeader.IntegrityCheck.Checksum.Header` is set to 0, as are the `Checksum.File` and `FileHeader.State`, prior to calculating (and setting) the checksum of the header. If the `FFS_ATTRIB_CHECKSUM` bit is set in the `FfsFileHeader.Attributes`, then the checksum for the remainder of the Ffs content must be generated and placed in the Checksum.File part of the `FfsFileHead.IntegrityCheck` structure.

The `FfsFileHeader.State` is zeroed, the `EFI_FILE_HEADER_CONSTRUCTION`, `EFI_FILE_HEADER_VALID` and `EFI_FILE_DATA_VALID` bits are set.

2.7.8.1 Reference Implementation: FDF file

The reference build uses the FDF file to specify construction of the FD, FVs and FFS files, as well as how to construct the different EFI Sections (what content is put into each section). Flags for attributes and alignment values are specified in the FDF file. These values are used to set the bits in FFS Header. As an example, if multiple sections are specified with different alignment values, only the maximum value of the alignment is used, and all sections are aligned to that value. Additionally, the sections are placed into the FFS in the order they appear in the FDF or specified by the Rules section of the FDF configuration file. Each driver is put into an FFS of its own. Also, EDK II expects the ordering of PEIM and DXE ffs files to start with a optional dependency section, followed by the PE32, user interface and finally the version sections.

2.7.9 APRIORI Files

At most, here can be at most one `APRIORI` file in a given firmware volume.

The PEI file, named by GUID of `PEI_APRIORI_FILE_NAME_GUID`, will specify the order of invocation of PEIMs by the PEI foundation. This is a special file, of the type, `EFI_FV_FILETYPE_FREEFORM` with a single `EFI_SECTION_RAW` and has the format:

```
typedef struct {
  EFI_GUID    FileNamesWithinVolume[NumberOfModulesInVolume];
} PEI_APRIORI_FILE_CONTENTS;
```

The DXE file, named by GUID of `DXE_APRIORI_FILE_NAME_GUID`, will specify the dispatch order of drivers by the DXE foundation. This is a special file, of the type `EFI_FV_FILETYPE_FREEFORM` with a single `EFI_SECTION_RAW` and has the same format as the `PEI_APRIORI_FILE_CONTENTS`.

2.7.10 Generating EFI Firmware Volume (FV) Files

This section provides the overview for generating an FV file, which contains an FV header and a sequence of FFS files. FVs are usually implemented so that the SEC and PEI Foundation are not compressed, while most PEIMs are executed from ROM (see above). As a result, these images are typically placed in a separate FV, with post-PEI phase modules placed in one or more FVs that are compressed. Reference Section 2.7.11 below.

The FV files are combinations of FFS files. For SEC, PEI Foundation and most PEIMs that execute directly from ROM, will need to have the BaseAddress re-based to the location of the driver in ROM. There are three different types of rebase actions. The first action is for the initial Boot drivers, while the most common is for execute in place (XIP) drivers. Some Runtime drivers may also need to be rebased. As part of the rebase these execute from ROM drivers may need to be aligned to the natural alignment of the machine architecture (or section alignment).

2.7.10.1 Reference Implementation: Combining FFS files into FV files

The reference build implementation uses the Firmware Description File (FDF) to describe how to combine FFS files into different FV files, as well as the layout of the FD files within an FD description. Each FV definition within the FDF is used to complete a data structure for constructing the FV. It should also be noted that the reference build does not use extended headers. The FvName in the FV_INFO structure is used to identify the name of the files that will be created in the FV directory.

```
typedef struct {
  EFI_PHYSICAL_ADDRESS     BaseAddress;
  EFI_PHYSICAL_ADDRESS     BootBaseAddress;
  EFI_PHYSICAL_ADDRESS     RuntimeBaseAddress;
  EFI_GUID                 FvGuid;
  UINTN                    Size;
  CHAR8                    FvName[_MAX_PATH];
  EFI_FV_BLOCK_MAP_ENTRY   FvBlocks[MAX_NUMBER_OF_FV_BLOCKS];
  EFI_FVB_ATTRIBUTES       FvAttributes;
  CHAR8                    FvFiles[MAX_NUMBER_OF_FILES_IN_FV][_MAX_PATH];
} FV_INFO;
```

The FV file header (see **EFI_FIRMWARE_VOLUME_HEADER** definition in Section 3) is constructed using the following information.

The first 16 bytes (*ZeroVector*) are set to zero. The FV **FileSystemGuid** is assigned a PI Specification defined GUID (**EFI_FIRMWARE_FILE_SYSTEM2_GUID**) that identifies it as a PI compliant Firmware Volume. The **Signature** is set to "**_FVH**" and the reserved byte is set to zero. The *PI Specification* defined **Revision** is set to 0x02.

As FFS files are added to the FV, the length of the FFS is added to the **FvLength** field, such that the final **FvLength** is complete length of the firmware volume, including the header (and extended header information). Also, as an FFS file is added to the FV, if the driver executes from ROM, the base address of the driver will be adjusted (rebased) within the FFS file to the physical location in ROM (BaseAddress + offset).

Attributes (defined in the FDF file) are set that define the capabilities and power-on defaults of this FV. These come from the FvAttributes of the FV_INFO data structure. The **HeaderLength** is set to size of header, including the size of the {0,0} terminated **BlockMap** data array.

The **BlockMap** data array is a mapping of the FFS files - giving the length (in blocks) and block size for each FFS file in the FV, starting with the first FFS file. This is an index of the blocks, and does not specify each FFS by name. If an extended header is required, it should be placed immediately following the **BlockMap** data array. The **ExtHeaderOffset** is set to the location of the extended header. Each block will be aligned on the largest value specified by the **EFI_FVB2_ALIGNMENT** attribute. Note that it is permissible to use variable block length devices, and as such, each block entry would have the

`BlockMap[index].NumBlocks` =1, while the `BlockMap[index].BlockLength` would vary according to the size of the FFS file (plus any padding value needed to align the next block on a natural boundary).

If the extended header is not included, then the `ExtHeaderOffset` is set to zero. If an extended header is present, it is followed by zero or more variable length extension (`EFI_FIRMWARE_VOLUME_EXT_ENTRY`) entries.

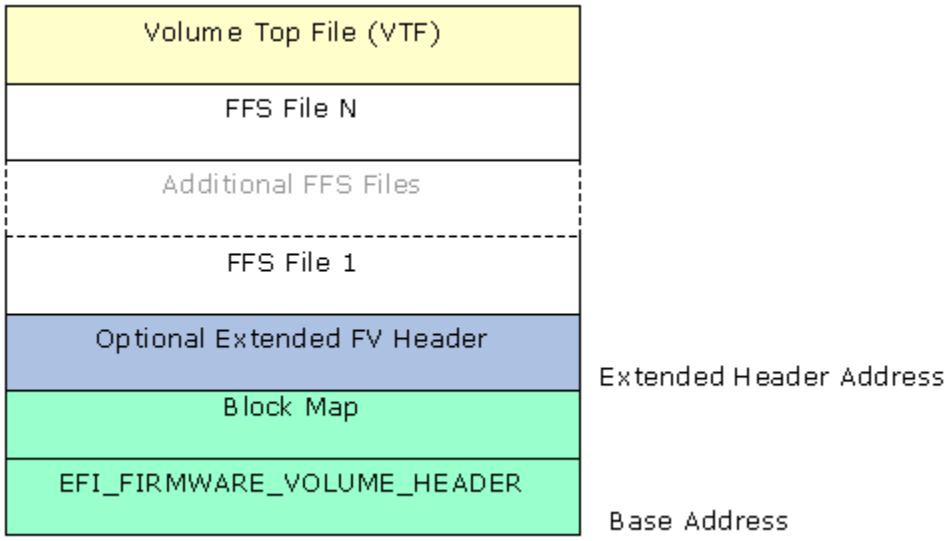


Figure 15. Firmware Volume Layout

Prior to adding the last FFS file (as specified in the FDF file), the FFS file must be modified to comply with the Volume Top File specification.

After the last FFS file has been added (so that the `FvLength` is complete), the `Checksum` field is set to zero and a checksum is calculated on the header so that a valid header will sum to zero (and placed into the `Checksum` field).

2.7.11 Implementing Compression

As stated in earlier sections, images that are executed from ROM may not be compressed. Images that contain .reloc sections, or that are executed post PEI phase may be compressed to save space in ROM. For best space savings and performance, an entire FV (containing all post PEI phase code) should be compressed, rather than compressing individual drivers. Decompression routines take a finite amount of time which can be additive - especially on copy commands - which can result in an unacceptable boot speed. Additionally, most compression algorithms provide better compression over larger data sets.

The EDK II Build system supports EFI standard compression as well as CRC32 validation within the tools.

Any other form of compression, encoding or signing must be through external tools. A TianoCompress tool is provided with the reference build.

The GUIDED encapsulation section method is used to control these additional tools. By definition (UEFI/PI specifications) a named GUID for a GUIDED encapsulation section is

used to provide information about how to process the section.

Assigning a GUID to a tool, such as TianoCompress, in the tools_def.txt file (refer to Section 5) the TianoCompress.exe application can be used to compress an EFI image - FV, FFS and/or SECTION. The following shows two lines that are in the tools_def.txt file to identify the TianoCompress tool.

```
*_MYTOOLS_*_TIANO_PATH  = TianoCompress.exe
*_MYTOOLS_*_TIANO_GUID  = A31280AD-481E-41B6-95E8-127F4C984779
```

2.7.11.1 Reference Implementation: GUIDED encapsulation sections

When the Build system encounters a GUIDED encapsulation section in the FDF file, the GUID is tested against GUIDs registered in the tools_def.txt file. If a match is found, then the executable tool associated in the GUID is executed on the encapsulated data defined in the FDF file. Since the tool is not present during a system boot, any optional tool must be able to provide code that can be used by any decompression, signing or verification drivers during boot. The following shows the use of the TianoCompress GUID in a sample FDF file for an FVMAIN image that contains all post-PEI modules.

```
FILE FV_IMAGE = E76CB2EC-A71A-42e8-8F34-56237870BD12 DEBUG_MYTOOLS_IPF {
       SECTION GUIDED A31280AD-481E-41B6-95E8-127F4C984779 {
          SECTION FV_IMAGE = FVMAIN
       }
}
```

In the example above, the first GUID (starting with E76CB2EC) is the EFI Name for the firmware volume, while the second (following the SECTION GUIDED statement) identifies the tool (TianoCompress.exe) that will be used on the FV section specified within the curly brackets after the GUID.

The provided reference build requires that all such tools take a minimum of three options on the command line. The -e option specifies that the tool will encrypt, compress, encode or sign the file specified on the command line. The -o option specifies the name of the output file to be created when using the -e option. A third option, -d, is used to decrypt, decompress, decode or verify the file specified on the command line.

Note: *Additional options may be included with the tool, however, the build system only requires these three options.*

2.7.12 Implementing Encryption or Signing

The same techniques used for implementing custom compression can be used for tools that are used for signing or encrypting an image.

2.7.13 Generating an FD image file

This section provides the overview for generating a platform flash file. When generating the FD file, the flash device is assumed to be "partitioned" into different areas, with the content of each added in sequence (with zero filled padding of any partial blocks to the next specified offset).

2.7.13.1 Reference Implementation: Data structures

The Reference Build system will create an FD file in the FV output directory, and using

the information in the FDF file, will add all FV files, as well as any data structures that are specified in the FDF file. Each FV will be added to the FD file in the order specified by the [FD] section at the location specified. Data structures in the FDF are typically used to initialize the data area for use by EFI drivers, and as such, may require an FV header to identify the region (such as NV storage) in Flash.

2.7.14 Generating Applications

This section provides and overview to generating UEFI applications which may or may not be resident in a flash device. Applications fall into three different types, applications that execute from within a flash image, applications that execute from the EFI Shell and applications that execute from an Operating System(accessing UEFI runtime services or need to access UEFI System Table fields). The build only provides support for UEFI applications that execute from within the flash image and applications that execute from the EFI Shell. These statically linked applications cannot make use of OS standard libraries or headers.

Applications that are executed within the flash image must be stored in an FFS file, along with the optional version and user interface sections. These applications are installed as part of the standard shell commands. The only way to execute a command that is executed within the flash image is to install it along with the shell commands. Otherwise, they cannot be executed.

Applications that execute from the EFI shell are PE32/COFF applications that have a modified header, and do not need to be placed within an FFS file. The .efi file generated by the $(MAKE) stage is capable of being executed from the shell command prompt.

2.7.15 Generating an Option ROM file

This section provides the overview for generating an external PCI OptionROM, where the driver is on a PCI add-in card. PCI devices that are laid down on a platform board, rather than on an add-in card (Ethernet, Video, Audio, etc. devices), will most likely have the driver resident in an FFS/FV/FD with the device vendor providing the driver code to the board vendor. A PCI Option ROM is typically discovered during system initialization, and the driver will be dispatched by the DXE Foundation. PCI Option ROM drivers are constructed from either EFI files or Binary files or a combination of both.

Most EFI implementations of PCI Option ROMs can use EFI compression for the driver, so that the ROM image fits into a smaller size device on the PCI add-in card. These drivers can use NV storage space in the primary on-board flash device, provided they register the system table data. If the driver is compressed, the size of the compressed file must be an even multiple of 512 bytes.

Note: *The maximum size for the driver code is 16MB, so drivers that are larger than 16MB must be compressed.*

The EFI file (PE32+ with modifications to the .data, .text, .reloc and .debug sections - see Generating Leaf EFI_SECTION Files above) will have an `EFI_PCI_EXPANSION_ROM_HEADER` prefixed to the EFI file (aligned on a 4-byte boundary). The header structure is defined in the PCI industry standard specification, and is shown below.

```
typedef struct {
  UINT16  Signature;
  UINT16  InitializationSize;
  UINT32  EfiSignature;
  UINT16  EfiSubsystem;
  UINT16  EfiMachineType;
  UINT16  CompressionType;
  UINT8   Reserved[8];
  UINT16  EfiImageHeaderOffset;
  UINT16  PcirOffset;
} EFI_PCI_EXPANSION_ROM_HEADER;
```

The *Signature* value of the PCI 3.0 version header is defined as **0xAA55**, and the EfiSignature is defined as **0x0EF1**. The **InitializationSize** is the number of 512-byte blocks of the driver image plus the size of this header. The *EfiSubsystem* is set to the value of PE32 Optional Header's Subsystem value, while the *EfiMachineType* is set to the **EFI_IMAGE_FILE_HEADER**'s Machine Type. The *CompressionType* field is set to either **0x0000** for no compression, or **0x0001** for standard EFI compression - no other compression types are permitted. The reserved bits are typically set to 0. However they may be used. The *EfiImageHeaderOffset* is set to the size of this header, while the *PcirOffset* is the offset to the EFI header, (the OptionROM header size plus any padding bytes to align the driver on its natural alignment boundary). Additionally, the PCI Data Structure (PCI 3.0 compliant is the default) is also inserted. The Vendor ID and Device ID are inserted into the PCI Data Structure. The ClassCode and CodeRevision are determined from the input file header information, while the ImageLength field is set to the OptionROM Header's InitializationSize field. All other fields are currently set to 0 by the reference implementation's EfiRom tool.

PCI device Expansion ROMs may contain code for multiple processor architectures. This may be implemented in a single physical ROM image, which can contain as many code images as desired for different system and processor architectures, Figure 16, below.

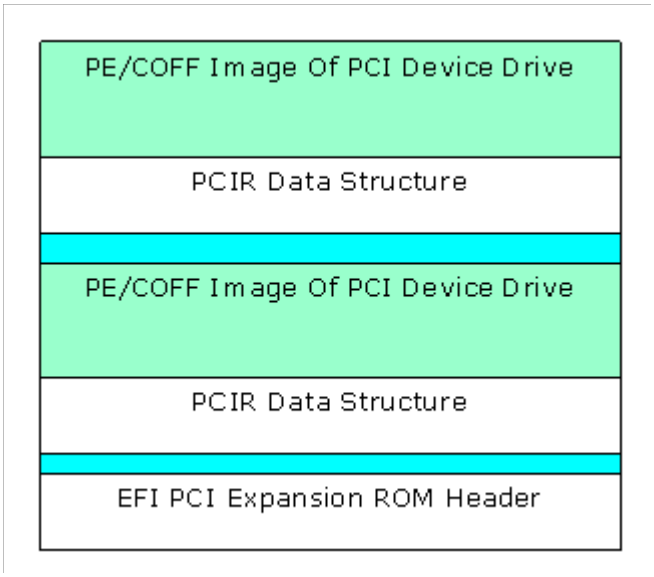

PE/COFF Image Of PCI Device Drive
PCIR Data Structure
PE/COFF Image Of PCI Device Drive
PCIR Data Structure
EFI PCI Expansion ROM Header

Figure 16. EFI PCI Expansion Option ROM layout

Each image must start on a 512-byte boundary and must contain the PCI Expansion ROM header. The starting point of each image depends on the size of previous images. The last image in a ROM has a special encoding in the header to identify it as the last image (*PCI Firmware Specification*, Revision 3.0).

Legacy Option ROM images must be the first image in the ROM image. The following is a list of the image combinations that may be placed in a PCI option ROM. This is not an exhaustive list. Instead, it provides what will likely be the most common PCI option ROM layouts. EFI complaint system firmware must work with all of these PCI option ROM layouts, plus any other layouts that are possible within the PCI Specification. The format of a Legacy Option ROM image is defined in the PCI Specification.

- Legacy Option ROM image
- Legacy Option ROM image + IA-32 EFI driver
- Legacy Option ROM image + Itanium Processor Family EFI driver
- Legacy Option ROM image + IA-32 EFI driver + Itanium Processor Family EFI driver
- Legacy Option ROM image + IA-32 EFI driver + x64 EFI driver
- Legacy Option ROM image + EBC Driver
- IA-32 UEFI driver
- Itanium Processor Family EFI driver
- IA-32 UEFI driver + Itanium Processor Family EFI driver
- EBC Driver

It is also possible to place an application in a PCI Option ROM. The exact mechanism by which applications can be loaded and executed from a PCI Option ROM is outside the scope of this document.

2.7.15.1 Reference Implementation: Binary Option ROM code

Pre-existing Binary OptionROM code can also be provided by hardware vendors. These images are verified to be of the correct format and length. If the length of the provided image is not an exact 512-byte multiple, padding bytes are added to ensure the image is an exact multiple of 512 bytes. If this occurs, a new checksum is calculated and replaces an existing checksum value.

2.7.16 Generating Capsule Update Files

This section provides the overview for generating Capsule files. Capsules are formatted variable length data structures that are passed from runtime back to the preboot phases (PEI, DXE, BDS). They are intended to be the major vehicle for delivering firmware volume changes. Capsules are constructed with a capsule header and the capsule volume. Content within the capsule volume usually includes a Firmware Volume as well as a Configuration Results (CR) file. The CR file is a string of Internal Forms Representation (IFR) name, value pairs as defined by the Human Interface Infrastructure (HII).

After identifying and creating the Firmware Volume that will be included in the capsule, the capsule header will be constructed. The header is constructed as follows.

The `CapsuleGuid` defines the format of the capsule data - including any optional header information. The format for a capsule is shown in Figure 15.

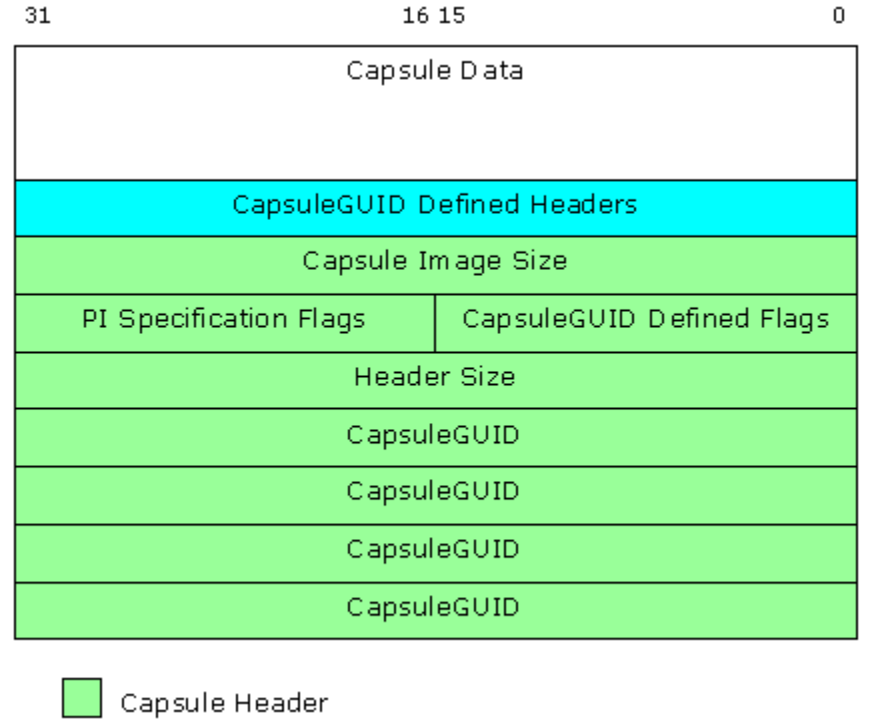


Figure 17. EFI Capsule Layout

2.7.16.1 Reference Implementation: Capsule generation and content

The reference build tool provides functionality to generate capsules. The capsule data content is defined in the FDF file for a given platform

3
UEFI and PI Image Specification

This section defines the data structures used for UEFI/PI images. This section is listed for completeness, however, as specifications change or errata are published, the actual data structure content may vary. Consult other industry standards for the latest versions and updates to UEFI 2.3 and PI 1.2 specifications.

EFI_TE_IMAGE_HEADER

Header format for Terse (TE) images

```
typedef struct {
  UINT16                    Signature;
  UINT16                    Machine;
  UINT8                     NumberOfSections;
  UINT8                     Subsystem;
  UINT16                    StrippedSize;
  UINT32                    AddressOfEntryPoint;
  UINT32                    BaseOfCode;
  UINT64                    ImageBase;
  EFI_IMAGE_DATA_DIRECTORY  DataDirectory[2];
} EFI_TE_IMAGE_HEADER;
```

Parameters

Signature

TE format = 0x5A56 ("VZ")

Machine

From the original file header:

0x014c	i386
0x0200	IPF
0x0EBC	EBC
0x08664	X64

NumberOfSections

1-byte packed value from the original file header

Subsystem

1-byte packed value from original optional header

StrippedSize

Original size minus the number of bytes we removed

AddressOfEntryPoint

Offset to entry point - from original optional header

BaseOfCode

From original image - required for ITP debug

ImageBase

From original file header

DataDirectory

Only two: the base relocation and debug directory

Related Definitions

```
typedef struct {
  UINT32  VirtualAddress;
  UINT32  Size;
} EFI_IMAGE_DATA_DIRECTORY;
```

EFI_IMAGE_DOS_HEADER

This is the DOS header of a PE/32+ exe or dll file.

```
typedef struct {
  UINT16  e_magic;
  UINT16  e_cblp;
  UINT16  e_cp;
  UINT16  e_crlc;
  UINT16  e_cparhdr;
  UINT16  e_minalloc;
  UINT16  e_maxalloc;
  UINT16  e_ss;
  UINT16  e_sp;
  UINT16  e_csum;
  UINT16  e_ip;
  UINT16  e_cs;
  UINT16  e_lfarlc;
  UINT16  e_ovno;
  UINT16  e_res[4];
  UINT16  e_oemid;
  UINT16  e_oeminfo;
  UINT16  e_res2[10];
  UINT32  e_lfanew;
} EFI_IMAGE_DOS_HEADER;
```

Parameters

e_magic
: The Magic number

e_cblp
: Bytes on the last page of file

e_cp
: *pages in file*

e_crlc
: Relocations

e_cparhdr
: Size of header in paragraphs

e_minalloc
: Minimum extra paragraphs needed

e_maxalloc
: Maximum extra paragraphs needed

e_ss
: Initial (relative) SS value

e_sp
: Initial SP value

e_csum
: Checksum

e_ip
: Initial IP value

e_cs
: Initial (relative) CS Value

e_lfarlc
: File address of relocation table

e_ovno
: Overlay number

e_res
: Reserved words

e_oemid
: OEM identifier (for e_oeminfo)

e_oeminfo
: OEM Information, e_oemid specific

e_res2
: Reserved words

e_lfanew
: File address of new .exe header

Related Definitions

```
union {
  EFI_IMAGE_OPTIONAL_HEADER32;
  EFI_IMAGE_OPTIONAL_HEADER64;
} EFI_IMAGE_OPTIONAL_HEADER;

typedef struct {
  // Common structure
  UINT16                    Magic;
  UINT8                     MajorLinkerVersion;
  UINT8                     MinorLinkerVersion;
  UINT32                    SizeOfCode;
  UINT32                    SizeOfInitializedData;
  UINT32                    SizeOfUninitializedData;
  UINT32                    AddressOfEntryPoint;
  UINT32                    BaseOfCode;
  // IA32/X64/EBC definitions follow
  UINT32                    BaseOfData;
  UINT32                    ImageBase;
  UINT32                    SectionAlignment;
  UINT32                    FileAlignment;
  UINT16                    MajorOperatingSystemVersion;
  UINT16                    MinorOperatingSystemVersion;
  UINT16                    MajorImageVersion;
  UINT16                    MinorImageVersion;
  UINT16                    MajorSubsystemVersion;
  UINT16                    MinorSubsystemVersion;
  UINT32                    Win32VersionValue;
  UINT32                    SizeOfImage;
  UINT32                    SizeOfHeaders;
  UINT32                    CheckSum;
  UINT16                    Subsystem;
  UINT16                    DllCharacteristics;
  UINT32                    SizeOfStackReserve;
  UINT32                    SizeOfStackCommit;
  UINT32                    SizeOfHeapReserve;
  UINT32                    SizeOfHeapCommit;
  UINT32                    LoaderFlags;
  UINT32                    NumberOfRvaAndSizes;
  EFI_IMAGE_DATA_DIRECTORY
DataDirectory[EFI_IMAGE_NUMBER_OF_DIRECTORY_ENTRIES];
} EFI_IMAGE_OPTIONAL_HEADER32;

typedef struct {
  // Common structure
  UINT16                    Magic;
  UINT8                     MajorLinkerVersion;
  UINT8                     MinorLinkerVersion;
  UINT32                    SizeOfCode;
```

```
  UINT32                  SizeOfInitializedData;
  UINT32                  SizeOfUninitializedData;
  UINT32                  AddressOfEntryPoint;
  UINT32                  BaseOfCode;
  // IPF Definitions follow
  UINT64                  ImageBase;
  UINT32                  SectionAlignment;
  UINT32                  FileAlignment;
  UINT16                  MajorOperatingSystemVersion;
  UINT16                  MinorOperatingSystemVersion;
  UINT16                  MajorImageVersion;
  UINT16                  MinorImageVersion;
  UINT16                  MajorSubsystemVersion;
  UINT16                  MinorSubsystemVersion;
  UINT32                  Win32VersionValue;
  UINT32                  SizeOfImage;
  UINT32                  SizeOfHeaders;
  UINT32                  CheckSum;
  UINT16                  Subsystem;
  UINT16                  DllCharacteristics;
  UINT64                  SizeOfStackReserve;
  UINT64                  SizeOfStackCommit;
  UINT64                  SizeOfHeapReserve;
  UINT64                  SizeOfHeapCommit;
  UINT32                  LoaderFlags;
  UINT32                  NumberOfRvaAndSizes;
  EFI_IMAGE_DATA_DIRECTORY
DataDirectory[EFI_IMAGE_NUMBER_OF_DIRECTORY_ENTRIES];
} EFI_IMAGE_OPTIONAL_HEADER64;
```

Related Definitions

Magic

For **EFI_IMAGE_OPTIONAL_HEADER32**, *Magic* **= IMAGE_NT_OPTIONAL_HDR32**, which is defined as: 0x10b.

For **EFI_IMAGE_OPPTIONAL_HEADER64**, *Magic* **= IMAGE_NT_OPTIONAL_HDR64**, which is defined as: 0x20b.

EFI_IMAGE_FILE_HEADER

This is the COFF header defined in the PE specification and is 20 bytes in length.

```
typedef struct {
  UINT16  Machine;
  UINT16  NumberOfSections;
  UINT32  TimeDateStamp;
  UINT32  PointerToSymbolTable;
  UINT32  NumberOfSymbols;
  UINT16  SizeOfOptionalHeader;
  UINT16  Characteristics;
} EFI_IMAGE_FILE_HEADER
```

Parameters

Machine

Subsystem type

NumberOfSections

The number of .text, .data, .rdata and .reloc sections in the image

TimeDateStamp

The time and date of image creation - typically during a platform build, all modules will have this field set to one common date and time.

PointerToSymbolTable

Pointer to the Symbol Table

NumberOfSymbols

Number of symbols

SizeOfOptionalHeader

Size of the optional header

Characteristics

File characteristics - the bits of this field are interpreted as follows.
Bit 0 - Relocations - if 1, then relocation information has been stripped.
Bit 1 - Executable - if set, there are no unresolved external references.
Bit 2 - Line Numbers - if set, line numbers have been stripped from the file.
Bit 3 - Local symbols - if set, local symbols have been stripped from the file.
Bits 4, 5 & 6 - reserved
Bit 7 - Machine Word - if set the bytes of machine word are reversed.
Bit 8 - 32-bit word machine - if set, target is a 32-bit OS.
Bit 9 - Debug - if set, debugging info stripped from the file in the .DBG file.
Bit 10 & 11 - reserved
Bit 12 - System File, if set, this is a system file.
Bit 13 - DLL, if set, this is a DLL.
Bit 14 - reserved.
Bit 15 - Machine Word - if set, bytes of machine word are reversed.

EFI_IMAGE_SECTION_HEADER

This replaces the .text and .data headers in PE32+ images.

```
typedef struct {
  UINT8 Name[EFI_IMAGE_SIZEOF_SHORT_NAME];
  union {
    UINT32  PhysicalAddress;
    UINT32  VirtualSize;
  } Misc;
  UINT32  VirtualAddress;
  UINT32  SizeOfRawData;
  UINT32  PointerToRawData;
  UINT32  PointerToRelocations;
  UINT32  PointerToLinenumbers;
  UINT16  NumberOfRelocations;
  UINT16  NumberOfLinenumbers;
  UINT32  Characteristics;
} EFI_IMAGE_SECTION_HEADER;
```

EFI_COMMON_SECTION_HEADER

This is the common definition for all EFI_SECTION headers. The EFI_COMMON_SECTION_HEADER is for sections less than 16MB in size, while the EFI_COMMON_SECTION_HEADER2 is for sections 16MB or larger. If the Size field is 0xFFFFFF, then the section header is the EFI_COMMON_SECTION_HEADER2.

```
typedef struct {
  UINT8             Size[3];
  EFI_SECTION_TYPE  Type;
} EFI_COMMON_SECTION_HEADER;

typedef struct {
  UINT8             Size[3];
  EFI_SECTION_TYPE  Type;
  UINT32            ExtendedSize;
} EFI_COMMON_SECTION_HEADER2;
```

Parameters

Size

A 24-bit unsigned integer that contains the total size of the section in bytes, including the **EFI_COMMON_SECTION_HEADER**. For example, a zero-length section has a Size of 4 bytes.

Type

One of the EFI_SECTION type 8-bit values.

EFI_VERSION_SECTION

This is the entire **EFI_VERSION_SECTION**. A version section is a leaf section that contains a numeric build number and an optional Unicode string that represents the file revision.

To facilitate versioning of PEIMs, DXE drivers, and other files, a version section may be included in a file. There must never be more than one version section contained within a file. EFI_VERSION_SECTION2 must be used if the section is 16MB or larger.

```
typedef struct {
  EFI_COMMON_SECTION_HEADER  CommonHeader;
  UINT16                     BuildNumber;
  CHAR16                     VersionString[1];
} EFI_VERSION_SECTION;

typedef struct {
  EFI_COMMON_SECTION_HEADER2 CommonHeader;
  UINT16                     BuildNumber;
  CHAR16                     VersionString[1];
} EFI_VERSION_SECTION2;
```

Parameters

BuildNumber

A UINT16 value that represents a particular build. Subsequent builds have monotonically increasing build numbers relative to earlier builds.

VersionString

The VersionString is a null terminated UNICODE string that contains a text representation of the version. If there is no text representation of the version, then an empty string must be provided.

CommonHeader

Common section header. CommonHeader.Type = **EFI_SECTION_VERSION**.

EFI_USER_INTERFACE_SECTION

This is the entire **EFI_USER_INTERFACE_SECTION**. The user interface file name section is a leaf section that contains a Unicode string that contains a human-readable file name.

This section is optional and is not required for any file types. There must never be more than one user interface file name section contained within a file. EFI_USER_INTERFACE_SECTION2 must be used if the section is 16MB or larger.

```
typedef struct {
  EFI_COMMON_SECTION_HEADER  CommonHeader;
  CHAR16                     FileNameString[...];
} EFI_USER_INTERFACE_SECTION;

typedef struct {
  EFI_COMMON_SECTION_HEADER2 CommonHeader;
  CHAR16                     FileNameString[...];
} EFI_USER_INTERFACE_SECTION2;
```

Parameters

FileNameString

The FileNameString is a UNICODE string that contains a human readable file name. This section is optional and is not required for any of the file types.

CommonHeader

Common section header. **CommonHeader.Type = EFI_SECTION_USER_INTERFACE.**

EFI_FREEFORM_SUBTYPE_GUID_SECTION

This is the entire **EFI_FREEFORM_SUBTYPE_GUID_SECTION**, where the GUID is the only content.

```
typedef struct {
  EFI_COMMON_SECTION_HEADER   CommonHeader;
  EFI_GUID                    SubTypeGuid;
} EFI_FREEFORM_SUBTYPE_GUID_SECTION;

typedef struct {
  EFI_COMMON_SECTION_HEADER2  CommonHeader;
  EFI_GUID                    SubTypeGuid;
} EFI_FREEFORM_SUBTYPE_GUID_SECTION2;
```

Parameters

SubTypeGuid

This GUID is defined by the creator of the file. It is a vendor-defined file type. Type **EFI_GUID** is defined in **InstallProtocolInterface()** in the *UEFI Specification, 2.0*.

CommonHeader

Common section header. **CommonHeader.Type = EFI_SECTION_GUID_DEFINED.**

EFI_COMPRESSION_SECTION

The header of an **EFI_COMPRESSION_SECTION**. A compression section is an encapsulation section in which the section data is compressed. To process the contents and extract the enclosed section stream, the section data must be decompressed using the decompressor indicated by the CompressionType parameter. The decompressed image is then interpreted as a section stream. EFI_COMPRESSION_SECTION2 is used if the section is 16MB or larger.

```
typedef struct {
  EFI_COMMON_SECTION_HEADER   CommonHeader;
  UINT32                      UncompressedLength;
  UINT8                       CompressionType;
} EFI_COMPRESSION_SECTION;

typedef struct {
  EFI_COMMON_SECTION_HEADER2  CommonHeader;
  UINT32                      UncompressedLength;
  UINT8                       CompressionType;
} EFI_COMPRESSION_SECTION2;
```

Parameters

UncompressedLength

A 24-bit unsigned integer that contains the total size of the section in bytes, including the **EFI_COMMON_SECTION_HEADER** after decompression.

CompressionType

0x01 for PI_STD compression, 0x00 if not compressed.

CommonHeader

Usual common section header. CommonHeader.Type = **EFI_SECTION_COMPRESSION**.

EFI_GUID_DEFINED_SECTION

The header of an **EFI_GUID_DEFINED_SECTION**. A GUID-defined section contains a section-type-specific header that contains an identifying GUID, followed by an arbitrary amount of data. It is an encapsulation section in which the method of encapsulation is defined by the GUID. A matching instance of EFI_GUIDED_SECTION_EXTRACTION_PROTOCOL (DXE) or EFI_GUIDED_SECTION_EXTRACTION_PPI (PEI) is required to extract the contents of this encapsulation section.

The GUID-defined section enables custom encapsulation section types for any purpose. One commonly expected use is creating an encapsulation section to enable a cryptographic authentication of the section contents. EFI_GUID_DEFINED_SECTION2 must be used if the section is 16MB or

larger.

```
typedef struct {
  EFI_COMMON_SECTION_HEADER   CommonHeader;
  EFI_GUID                    SectionDefinitionGuid;
  UINT16                      DataOffset;
  UINT16                      Attributes;
} EFI_GUID_DEFINED_SECTION;

typedef struct {
  EFI_COMMON_SECTION_HEADER2  CommonHeader;
  EFI_GUID                    SectionDefinitionGuid;
  UINT16                      DataOffset;
  UINT16                      Attributes;
} EFI_GUID_DEFINED_SECTION2;
```

Parameters

SectionDefinitionGuid

A GUID that defines the format of the data that follows. It is a vendor-defined format.

DataOffset

Contains the offset in bytes from the beginning of the common header to the first byte of the data.

Attributes

Bit field that declares some specific characteristics of the section contents. The bits are defined in "Related Definitions" below. If the bit is not defined below, it is reserved and must be set to zero.

CommonHeader

Usual common section header. **CommonHeader.Type = EFI_SECTION_GUID_DEFINED**.

Related Definitions

```
Bit 0    EFI_GUIDED_SECTION_PROCESSING_REQUIRED
Bit 1    EFI_GUIDED_SECTION_AUTH_STATUS_VALID
```

Table 8. Descriptions of Bit Fields for Attributes

Field	Description
EFI_GUIDED_SECTION_ PROCESSING_REQUIRED	Set to 1 if the section requires processing to obtain meaningful data from the section contents. Processing would be required, for example, if the section contents were encrypted or compressed. If the **EFI_GUIDED_SECTION_PROCESSING_REQUIRED** bit is cleared to zero, it is possible to retrieve the section's contents without processing in the absence of an associated instance of the **EFI_GUIDED_SECTION_EXTRACTION_PROTOCOL** (DXE) or **EFI_PEI_GUIDED_SECTION_EXTRACTION_PPI** (PEI).. In this case, the beginning of the encapsulated section stream is indicated by the value of **DataOffset**.
EFI_GUIDED_SECTION_ AUTH_STATUS_VALID	Set to 1 if the section contains authentication data that is reported through the *AuthenticationStatus* parameter returned from the **GUIDed Section Extraction Protocol.** If the **EFI_GUIDED_SECTION_AUTH_STATUS_VALID** bit is clear, the *AuthenticationStatus* parameter is not used.

Example

```
typedef struct {
  EFI_GUID_DEFINED_SECTION  GuidSectionHeader;
  UINT32                    CRC32Checksum;
} CRC32_SECTION_HEADER;
```

EFI_FFS_FILE_HEADER

The header of an FFS file.

```
typedef struct {
  EFI_GUID                  Name;
  EFI_FFS_INTEGRITY_CHECK   IntegrityCheck;
  EFI_FV_FILETYPE           Type;
  EFI_FFS_FILE_ATTRIBUTES   Attributes;
  UINT8                     Size[3];
  EFI_FFS_FILE_STATE        State;
} EFI_FFS_FILE_HEADER;

typedef struct {
  EFI_GUID                  Name;
  EFI_FFS_INTEGRITY_CHECK   IntegrityCheck;
  EFI_FV_FILETYPE           Type;
  EFI_FFS_FILE_ATTRIBUTES   Attributes;
  UINT8                     Size[3];
  EFI_FFS_FILE_STATE        State;
  UNIT32                    ExtendedSize;
} EFI_FFS_FILE_HEADER2;
```

Parameters

Name

This GUID is the file name. It is used to uniquely identify the file. There may be only one instance of a file with the file name GUID of Name in any given firmware volume.

IntegrityCheck

Used to verify the integrity of the file. Type **EFI_FFS_INTEGRITY_CHECK** is defined in "Related Definitions" below.

Type

Identifies the type of file. Type **EFI_FV_FILETYPE** is defined as a **UINT8**, following the coding values Defined FV File Types table above.

Attributes

Declares various file attribute bits. Type **EFI_FFS_FILE_ATTRIBUTES** is defined as:

Bit 0, 1 and 7 are reserved and must be set to 0.

Bit 2, is the FIXED bit, if set, the file may not be moved from its present location - XIP.

Bits 3,4 and 5 are the Data Alignment bits.

Bit 6 is the Checksum bit. If set to 0, then the Integrity.Checksum.File must initialized to 0x55AA, otherwise if it is set to 1, then the IntegrityCheck.Checksum.File is an 8-bit checksum of the entire file.

Size

The length of the file in bytes, including the FFS header. The length of the file data is either (Size - sizeof(**EFI_FFS_FILE_HEADER**)). This calculation means a zero-length file has a Size of 24 bytes, which is sizeof(**EFI_FFS_FILE_HEADER**). Size is not required to be a multiple of 8 bytes. Given a file F, the next file header is located at the next 8-byte aligned firmware volume offset following the last byte of the file F. If the value of **Size** equals, 0xFFFFFF, then the actual size is specified in the ExtendedSize field, and the header is **EFI_FFS_FILE_HEADER2**.

State

Used to track the state of the file throughout the life of the file from creation to deletion. Type EFI_FFS_FILE_STATE is a UINT8 with the following bit fields.

bit 0 - **EFI_FILE_HEADER_CONSTRUCTION**

bit 1 - **EFI_FILE_HEADER_VALID**

bit 2 - **EFI_FILE_DATA_VALID**

bit 3 - **EFI_FILE_MARKED_FOR_UPDATE**

bit 4 - **EFI_FILE_DELETED**

bit 5 - **EFI_FILE_HEADER_INVALID**

bits 6 & 7 are reserved and must be set to the ERASE_POLARITY defined in the **EFI_FIRMWARE_VOLUME_HEADER**.

Related Definitions

```
//*
// EFI_FFS_INTEGRITY_CHECK
//*
typedef union {
  struct {
    UINT8         Header;
    UINT8         File;
  } Checksum;
  UINT16          Checksum16;
} EFI_FFS_INTEGRITY_CHECK;

//*
// EFI_FV_FILETYPE
//*
typedef UINT8     EFI_FV_FILETYPE;

//*
// EFI_FFS_ATTRIBUTES
//*
typedef UINT8     EFI_FFS_FILE_ATTRIBUTES;
```

EFI_FIRMWARE_VOLUME_HEADER

The header of an FV file.

```
typedef struct {
  UINT8                    ZeroVector[16];
  EFI_GUID                 FileSystemGuid;
  UINT64                   FvLength;
  UINT32                   Signature;
  EFI_FVB_ATTRIBUTES_2     Attributes;
  UINT16                   HeaderLength;
  UINT16                   Checksum;
  UINT16                   ExtHeaderOffset;
  UINT8                    Reserved[1];
  UINT8                    Revision;
  EFI_FV_BLOCK_MAP         BlockMap[];
} EFI_FIRMWARE_VOLUME_HEADER;
```

Parameters

ZeroVector

The first 16 bytes are reserved to allow for the reset vector of processors whose reset vector is at address 0.

FileSystemGuid

Declares the file system with which the firmware volume is formatted. Type EFI_GUID is defined in InstallProtocolInterface() in the Unified Extensible Firmware Interface Specification, version 2.0 (UEFI 2.0 specification).

FvLength

Length in bytes of the complete firmware volume, including the header.

Signature

Set to {'_','F','V','H'}.

Attributes

Declares capabilities and power-on defaults for the firmware volume. Current state is determined using the GetAttributes() function and is not maintained in the Attributes field of the firmware volume header. Type EFI_FVB_ATTRIBUTES is UINT32 which is a bit encoded field (see Related Definitions" below).

HeaderLength

Length in bytes of the complete firmware volume header.

Checksum

A 16-bit checksum of the firmware volume header. A valid header sums to zero.

ExtHeaderOffset

Offset, relative to the start of the header, of the extended header (EFI_FIRMWARE_VOLUME_EXT_HEADER) or zero if there is no extended header.

The extended header is followed by zero or more variable length extension entries. Each extension entry is prefixed with the EFI_FIRMWARE_VOLUME_EXT_ENTRY structure, which defines the type and size of the extension entry. The extended header is always 32-bit aligned relative to the start of the file header.

Reserved

In this version of the specification, this field must always be set to zero.

Revision

Set to 2. Future versions of this specification may define new header fields and will increment the Revision field accordingly.

FvBlockMap[]

An array of run-length encoded FvBlockMapEntry structures. The array is terminated with an entry of {0,0}.

FvBlockMapEntry.NumBlocks

The number of sequential blocks in the run which are of the same size.

FvBlockMapEntry.BlockLength

The length of each block in the run.

Related Definitions

```
EFI_FVB_ATTRIBUTES_2
  // Attributes bit definitions
  #define EFI_FVB2_READ_DISABLED_CAP  0x00000001
  #define EFI_FVB2_READ_ENABLED_CAP   0x00000002
  #define EFI_FVB2_READ_STATUS        0x00000004
  #define EFI_FVB2_WRITE_DISABLED_CAP 0x00000008
  #define EFI_FVB2_WRITE_ENABLED_CAP  0x00000010
  #define EFI_FVB2_WRITE_STATUS       0x00000020
  #define EFI_FVB2_LOCK_CAP           0x00000040
  #define EFI_FVB2_LOCK_STATUS        0x00000080
  #define EFI_FVB2_STICKY_WRITE       0x00000200
  #define EFI_FVB2_MEMORY_MAPPED      0x00000400
  #define EFI_FVB2_ERASE_POLARITY     0x00000800
  #define EFI_FVB2_READ_LOCK_CAP      0x00001000
  #define EFI_FVB2_READ_LOCK_STATUS   0x00002000
  #define EFI_FVB2_WRITE_LOCK_CAP     0x00004000
  #define EFI_FVB2_WRITE_LOCK_STATUS  0x00008000
```

```
// Alignment Attribute bit definitions
#define EFI_FVB2_ALIGNMENT          0x001F0000
#define EFI_FVB2_ALIGNMENT_1        0x00000000
#define EFI_FVB2_ALIGNMENT_2        0x00010000
#define EFI_FVB2_ALIGNMENT_4        0x00020000
#define EFI_FVB2_ALIGNMENT_8        0x00030000
#define EFI_FVB2_ALIGNMENT_16       0x00040000
#define EFI_FVB2_ALIGNMENT_32       0x00050000
#define EFI_FVB2_ALIGNMENT_64       0x00060000
#define EFI_FVB2_ALIGNMENT_128      0x00070000
#define EFI_FVB2_ALIGNMENT_256      0x00080000
#define EFI_FVB2_ALIGNMENT_512      0x00090000
#define EFI_FVB2_ALIGNMENT_1K       0x000A0000
#define EFI_FVB2_ALIGNMENT_2K       0x000B0000
#define EFI_FVB2_ALIGNMENT_4K       0x000C0000
#define EFI_FVB2_ALIGNMENT_8K       0x000D0000
#define EFI_FVB2_ALIGNMENT_16K      0x000E0000
#define EFI_FVB2_ALIGNMENT_32K      0x000F0000
#define EFI_FVB2_ALIGNMENT_64K      0x00100000
#define EFI_FVB2_ALIGNMENT_128K     0x00110000
#define EFI_FVB2_ALIGNMENT_256K     0x00120000
#define EFI_FVB2_ALIGNMNET_512K     0x00130000
#define EFI_FVB2_ALIGNMENT_1M       0x00140000
#define EFI_FVB2_ALIGNMENT_2M       0x00150000
#define EFI_FVB2_ALIGNMENT_4M       0x00160000
#define EFI_FVB2_ALIGNMENT_8M       0x00170000
#define EFI_FVB2_ALIGNMENT_16M      0x00180000
#define EFI_FVB2_ALIGNMENT_32M      0x00190000
#define EFI_FVB2_ALIGNMENT_64M      0x001A0000
#define EFI_FVB2_ALIGNMENT_128M     0x001B0000
#define EFI_FVB2_ALIGNMENT_256M     0x001C0000
#define EFI_FVB2_ALIGNMENT_512M     0x001D0000
#define EFI_FVB2_ALIGNMENT_1G       0x001E0000
#define EFI_FVB2_ALIGNMENT_2G       0x001F0000
```

EFI_FIRMWARE_VOLUME_EXT_HEADER

The extended header is always 32-bit aligned relative to the start of the file header.

```
typedef struct {
  EFI_GUID  FvName;
  UINT32    ExtHeaderSize;
} EFI_FIRMWARE_VOLUME_EXT_HEADER;
```

Parameters

FvName

Firmware volume name.

ExtHeaderSize

Size of the rest of the extension header, including this structure.

EFI_FIRMWARE_VOLUME_EXT_ENTRY

After the extension header, there is an array of variable-length extension header entries, each prefixed with the **EFI_FIRMWARE_VOLUME_EXT_ENTRY** structure.

The EDK II build tools will always append either the content of a binary file or convert a user specified (FDF file) data array to binary for the content following this entry.

The build tools must calculate the ExtEntrySize and use the TYPE value specified in the FDF file.

```
typedef struct {
  UINT16  ExtEntrySize;
  UINT16  ExtEntryType;
} EFI_FIRMWARE_VOLUME_EXT_ENTRY;
```

Parameters

ExtEntrySize

Size of this header extension.

ExtEntryType

Type of the header. See **EFI_FV_EXT_TYPE_x**.

EFI_FIRMWARE_VOLUME_EXT_TYPE_OEM_TYPE

The EDK II build tool do not currently support generating this information. A user wishing to use this structure must construct either a binary file or specify an data array completing all of this information.

```
typedef struct {
  EFI_FIRMWARE_VOLUME_EXT_ENTRY  Hdr;
  UINT32                         TypeMask;
  EFI_GUID                       Types[];
} EFI_FIRMWARE_VOLUME_EXT_ENTRY_OEM_TYPE;
```

Parameters

Hdr

Standard extension entry, with the type **EFI_FV_EXT_TYPE_OEM_TYPE**.

TypeMask

A bit mask, one bit for each file type between 0xC0 (bit 0) and 0xDF (bit 31). If a bit is '1', then the GUID entry exists in Types. If a bit is '0' then no GUID entry exists in Types. For example, the value 0x01010301 would indicate that there would be five total entries in Types for file types 0xC0 (bit 0), 0xC8 (bit 4), 0xC9 (bit 5), 0xD0 (bit 16), and 0xD8 (bit 24).

Types

An array of GUIDs, each GUID representing an OEM file type. This extension header provides a mapping between a GUID and an OEM file type.

EFI_CAPSULE_HEADER

The following data structure is for an EFI Capsule Header. A capsule is simply a contiguous set of data that starts with an EFI_CAPSULE_HEADER. The CapsuleGuid field in the header defines the format of the capsule.

The capsule contents are designed to be communicated from an OS-present environment to the system firmware. To allow capsules to persist across system reset, a level of indirection is required for the description of a capsule, since the OS primarily uses virtual memory and the firmware at boot time uses physical memory.

```
typedef struct {
  EFI_GUID  CapsuleGuid;
  UINT32    HeaderSize;
  UINT32    Flags;
  UINT32    CapsuleImageSize;
} EFI_CAPSULE_HEADER;
```

Parameters

CapusuleGuid

A GUID that defines the content of a capsule.

HeaderSize

The size of the header (may be larger than the size of the **EFI_CAPSULE_HEADER**, as the GUID may imply extended header entries).

Flags

A bit-encode field describing the capsule's attributes. All undefined bits must be set to zero. Currently, only bits 16, 17 and 18 are defined - Flags values 0x0000 - 0xFFFF are defined by the CapsuleGuid. Values 0x00010000 - 0xFFFF0000 are defined by in the UEFI specification.

0x00010000 - CAPSULE_FLAGS_PERSIST_ACROSS_RESET
0x00020000 - CAPSULE_FLAGS_POPULATE_SYSTEM_TABLE
0x00040000 - CAPSULE_FLAGS_INITIATE_RESET

Note: *Note: A capsule which has the* **CAPSULE_FLAGS_INITIATE_RESET** *Flag must have* **CAPSULE_FLAGS_PERSIST_ACROSS_RESET** *set in its header as well. Firmware that encounters a capsule which has the* **CAPSULE_FLAGS_INITIATE_RESET** *Flag set in its header will initiate a reset of the platform which is compatible with the passed-in capsule request and will not return back to the caller.*

CapsuleImageSize

The length in bytes of the entire image (including all headers). If this value is greater than the size of the current file, the capsule was split into more than one file.

4
Reference Build Process Overview

The EDK II reference build process provides backward compatibility for existing EDK components and platforms, using EDK processes and tools will not be described in this document. The EDK build tools are provided as part of a compatibility package, or existing EDK trees, containing the EDK build tools. Some EDK components may also be built using the EDK II build tools, where those components are included in an EDK II platform. The exact list of EDK components is not provided here - other EDK II documentation contains information on using EDK components with EDK II.

The EDK II Build Process is handled in three major stages:

- AutoGen stage: parse meta-data files to generate some C source code files and the makefiles.
- $(MAKE) stage: process source code files to create PE32/PE32+/COFF images that are processed to EFI format using NMAKE (Microsoft operating system development platforms) or MAKE (for UNIX style operating system development platforms).
- ImageGen stage: takes the EFI format files and creates EFI "FLASH" images, UEFI Applications or EFI PCI Option ROMs.

Figure 18 shows the relationship of these three stages.

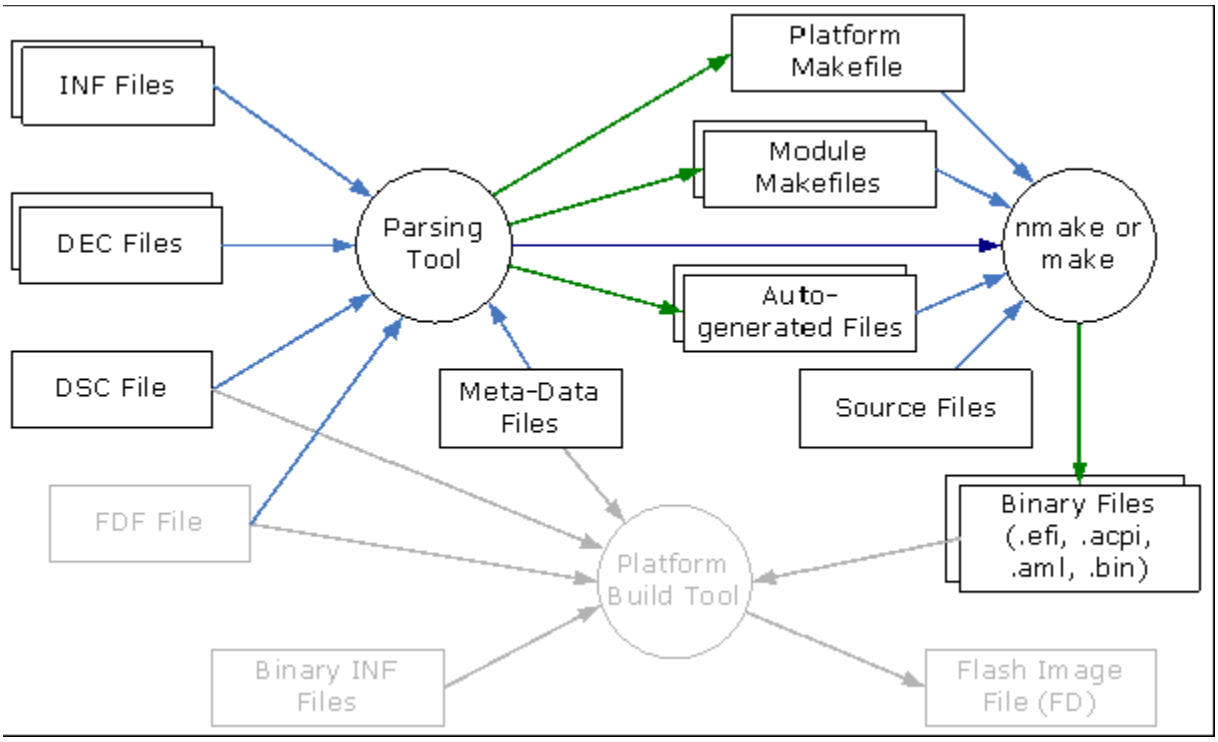


Figure 18. EDK II Build Process Flow

Note: *In Figure 18, Meta-Data Files indicates build tool meta-data files: build_rule.txt, tools_def.txt, target.txt and the like.*

The Build process is organized so that if a FLASH image file is not required, such as in generating a Binary Module that will be distributed to other end-users, stage three can be skipped. Option ROM and/or UEFI applications can also be distributed in this fashion.

Note: *The Nt32Pkg (Nt32Pkg/Nt32Pkg.dsc) emulation platform requires Windows header files. In order to include Windows header files, execute the edk2setup.bat utility with the --nt32 option. This option should detect the Microsoft Visual Studio installation and will execute it's setup command, for example, vsvars32.bat.*

4.1 Supported Development Environments

The reference build system supports development on the following operating systems running on IA32 or X64 platforms. IPF development platforms are not supported (IPF targets are supported using cross-compiler features).

Microsoft Windows 2003 Server, Microsoft WindowsXP Professional, Microsoft Windows Vista, Red Hat Enterprise Linux 4 Update 5 (Server and Desktop), SuSE Linux Enterprise 10 (Server and Desktop) and Apple Mac OS/X 10.4.x (running on iMac hardware).

Note: *Microsoft compiler tools are 32-bit applications. Running 32-bit applications on a 64-bit Windows operating system is possible, but the application will run up to 4 times slower than if it were run on a system with a 32-bit Windows operating system. Installing a Windows 32-bit operating system on a 64-bit system is also permitted, and no performance degradation has been seen using this configuration.*

4.2 Supported Development Tools

Table 9 lists the Third-Party Compiler Tool chains supported under the reference build.

Note: *The GCC compiler tool chains should be built using provided scripts for generating the tools.*

Table 9. Supported Compiler Tools

Vendor	Target Architecture	Tool	Version
Microsoft	IA32/X64/IPF	Visual Studio Team Suite	2005
Microsoft	IA32/X64/IPF	Visual Studio Team Suite	2008
Microsoft	IA32/X64/IPF	Driver Development Kit (DDK)	3790.1830
Intel	IA32/X64/IPF	Intel C++ Compiler for Windows	11.1
Intel	IA32/X64/IPF	Intel C++ Compiler for Linux	11.1
Intel	IA32/X64/IPF	Intel C++ Compiler for Mac OS/X	11.1
Intel	EBC	Intel C Compiler for EFI Byte Code	1.3
GCC	IA32/X64/IPF	GCC	4.4

Additional tools may be required for platform development.

Table 10 lists additional supported tools that may be required.

Note: *How to use the lzma tool is not included in this document, however, it is a well known compression algorithm and is included in the table for reference.*

Table 10. Additional Compiler Tools

Vendor	Tool Name	Version	Description
Microsoft	asl.exe	3.0.0NT	ACPI Source Language Assembler (May 3, 2006)
ACPICA	iasl.exe	20060113	ASL Optimizing Compiler
7-zip	lzma.exe	4.06	LZMA compression tool

Note: *Install the tool chains for compilers and/or additional tools prior to building any image.*

4.3 GCC Setup

Unlike Microsoft and Intel compiler tools that are distributed with installation software, the GCC tool chains can be pulled directly from the Internet. Since different Linux distributions and OS/X installations may use different (and possibly incompatible) versions, testing of EDK II with GCC is limited to specific releases. To ensure UEFI/PI compliant GCC tools, scripts have been provided in the BaseTools/gcc folder of the BaseTools source tree (part of buildtools.tianocore.org). The GCC tools built by these tools are the only recommended GCC tool suite for creating UEFI/PI images.

Note: *Although your OS distribution may bundle a MinGW based GCC/binutils, these are not supported & validated for UEFI/PI compatibility. Please refer to the README file under BaseTools/gcc for more information on building the GCC tool suite.*

After building the gcc cross-compiler tool chain, it may be necessary to modify Conf/ tools_def.txt to configure the UNIXGCC tool chain paths.

The released tools' source code is provided as part of the EDK II project in the BaseTools/Source directory tree.

The Sources for the build tools is available via Subversion in the BuildTools project on TianoCore.org. Build tools are written in either C or Python - tools that processes binary data are written in C, while tools that process text files are written in Python.

Note: *It is not required to download the tool sources from the BuildTools project for systems running 32-bit versions of the Microsoft Operating systems - Win32 binaries for the build tools are provided as part of the EDK II project.*

4.4 Build Specific Meta-Data Files

This section provides an overview of the meta-data files that are used to control different aspects of the build. There are three files of interest, build_rule.txt, tools_def.txt and target.txt.

The build_rule.txt file describes how different file types are processed, while tools_def.txt describes the location of third party tools and tool flags. The target.txt file is used as a filter, allowing a developer to build items of interest without having to build everything.

4.4.1 Parse Meta-Data - AutoGen stage

Additional module and platform meta-data in the EDK II reference build system is stored in text-based, INI-style, documents. Refer to the *EDK II INF Specification*, *EDK II DSC Specification*, *EDK II DEC Specification,* and *EDK II Flash Description File Specification* to see the format of these files.

Note: *More than one INF file may exist in a module directory, but the basename and GUID for these INF files must be different.*

The parse stage creates individual module and library autogen.c, autogen.h andmakefiles, as well as a single, top-level makefile. Since EDK II supports Microsoft, Intel and GCC complier tool chains, the Microsoft Build Tool, NMAKE/Makefile is for Windows developer Workstations using Microsoft or Intel tool chains on a Microsoft Windows operating system development workstation. For UNIX based development workstations, the GCC build tool, GMAKE/GNUmakefile is used.

Parse the target.txt and tools_def.txt files to build a list of compiler tools that will be used in makefiles, as well as to determine what modules will be built. All third party tools and flags for those tools get expanded in the generated makefiles. The following is an example of makefile statements that support this mode.

```
...
PP = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe
CC = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe
APP = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe
VFRPP = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe
DLINK = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\link.exe
PCH = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\cl.exe
ASM = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\ml.exe
TIANO = TianoCompress.exe
SLINK = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\lib.exe
ASMLINK = C:\WINDDK\3790.1830\bin\bin16\link.exe
ASL = C:\ASL\iasl.exe
...
DEFAULT_PP_FLAGS = /nologo /E /TC /FI$(DEST_DIR_DEBUG)/AutoGen.h
DEFAULT_CC_FLAGS = /nologo /W4 /WX /Gy /c /D UNICODE /O1ib2 /GL /
FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF /Gs8192 /Zi /Gm
DEFAULT_APP_FLAGS = /nologo /E /TC
DEFAULT_VFRPP_FLAGS = /nologo /E /TC /DVFRCOMPILE /FIAutoGen.h
DEFAULT_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /OPT:ICF=10 /
MAP /ALIGN:32 /MACHINE:I386 /LTCG /DLL /ENTRY:$(ENTRYPOINT) /SUBSYSTEM:CONSOLE /
SAFESEH:NO /BASE:0 /DRIVER /DEBUG /PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb
DEFAULT_PCH_FLAGS = /nologo /W4 /WX /Gy /c /D UNICODE /O1ib2 /GL /
FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF /Gs8192 /Fp$(DEST_DIR_OUTPUT)/
AutoGen.h.gch /Yc /TC /Zi /Gm
DEFAULT_ASM_FLAGS = /nologo /W3 /WX /c /coff /Cx /Zd /W0 /Zi
DEFAULT_TIANO_FLAGS =
DEFAULT_SLINK_FLAGS = /nologo /LTCG
DEFAULT_ASMLINK_FLAGS = /link /nologo /tiny
DEFAULT_ASL_FLAGS =
...
```

(continued next page)

```
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj : $(COMMON_DEPS)
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj :
$(WORKSPACE)\MdePkg\Include\Library\DebugLib.h
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj : $(WORKSPACE)\MdePkg\Include\Library\BaseLib.h
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj :
$(WORKSPACE)\MdePkg\Include\Library\TimerLib.h
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj :
$(WORKSPACE)\MdePkg\Include\Library\BaseMemoryLib.h
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj : $(WORKSPACE)\MdePkg\Include\Library\PcdLib.h
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj :
$(WORKSPACE)\MdePkg\Library\BaseLib\Ia32\WriteGdtr.c
$(OUTPUT_DIR)\Ia32\WriteGdtr.obj :
$(WORKSPACE)\MdePkg\Library\BaseLib\BaseLibInternals.h
  "$(CC)" /Fo$(OUTPUT_DIR)\Ia32\WriteGdtr.obj $(CC_FLAGS) $(INC)
$(WORKSPACE)\MdePkg\Library\BaseLib\Ia32\WriteGdtr.c

$(OUTPUT_DIR)\Ia32\WriteDr3.obj : $(COMMON_DEPS)
$(OUTPUT_DIR)\Ia32\WriteDr3.obj :
$(WORKSPACE)\MdePkg\Library\BaseLib\Ia32\WriteDr3.c
  "$(CC)" /Fo$(OUTPUT_DIR)\Ia32\WriteDr3.obj $(CC_FLAGS) $(INC)
$(WORKSPACE)\MdePkg\Library\BaseLib\Ia32\WriteDr3.c

  ...
```

One makefile is generated for each combination of TARGET_TOOLCHAIN and ARCH. The build_rule.txt file specifies how files are to be processed, with the generated makefiles containing valid syntax for the given tool chain.

4.4.1.1 Parse DSC file

1. Obtain platform Feature Flag PCD values may be used in the Conditional Directives - if the value of a Feature Flag PCD used in the conditional directives cannot be determined, the build will break.
2. Second pass over the DSC files will crush out any conditional directives where the feature flag expression used in the conditional directive is False.
3. Obtain platform PCD values which will go into the individual module AutoGen.h files where needed.
4. Obtain the FDF filename and obtain the Flash related PCDs from the FDF file.
5. Create the list of EDK Libraries that will need to be built
   a Create the makefiles
   b Add the makefile to the top level makefile
6. Create the list of EDK Components that will need to be built
   a Create the makefiles
   b Add the makefile to the top level makefile
7. Create the list of EDK II Modules that will need to be built
8. For each EDK II Module, parse the Module's INF file
   a Create a directed graph list of the EDK II Library Instances that will be use for the module.
   - Individual modules may require different compilation options, over-riding any global definitions.

   b Create the Library Instance's AutoGen.c files containing PCD, Guid, Protocol, Ppi and EntryPoint definitions and data structures.
   c Create the module's library instance makefiles

d Create the AutoGen.c files containing PCD, Guid, Protocol, Ppi and EntryPoint definitions and data structures.

e Create any Strings header file required for VFR processing.

- The VFR file name can not be same as a C file name in a module directory. If so, the same output files will be generated and overwritten. (A.vfr --> A.c --> A.obj, A.c --> A.obj.)

f Need the EDK II Build Spec to add this warning.

g Create the module's makefiles

h Add the makefile to the top level makefile

Add a last call in the top-level makefile to build the GenFds program only if an FDF file was specified in the DSC file.

4.4.2 Build Binary EFI Images - $(MAKE) stage

Binary EFI images are created in two steps; the first step uses 3rd party assemblers, compilers and linkers to generate a PE32/PE32+/COFF image file, while the second step uses the GenFw application provided in EDK II to modify the PE32/PE32+/COFF image file to create an EFI file with an EFI_IMAGE_SECTION_HEADER structure. Since different EFI SECTION types may require different values for alignment offsets, the GenFw tool must specify the component type, which is derived from the INF meta-data's ModuleType statement.

This stage is executed via either the NMAKE or the GMAKE build tool. The makefiles (Makefile or GNUMakefile) are created during the meta-data processing stage. The makefiles specify the compiler, linker, assembler, and GenFw tool commands and options. The Makefile (or GNUmakefile), which is a list of other NMAKE or GMAKE commands, will call the GenFds application to initiate the third stage of a build, IFF there is a Flash Definition File (FDF) specified in the DSC file. If no FDF file is specified, then the build will terminate with the creation of individual module EFI formatted (EFI) images.

4.4.3 Assemble FLASH Images - ImageGen stage

Once all the modules' EFI image files have been created, the final stage of a build process is called. For FLASH images, this stage uses the FDF file, and some parts of the DSC file to create the final binary image files. This stage processes the individual EFI files, formatting them into leaf EFI SECTION types and combining them using implied rules (or custom rules) into different firmware files (FFS), firmware volumes (FVs) and the final FLASH images (FDs). The construction of these images is based on the content of the FDF file (with a very limited amount of data being obtained from the DSC file).

The default build rules specify removal of .reloc sections of the PE32/PE32+ file for all SEC, PEI_CORE and PEIM modules and components. To prevent removal of the .reloc section, a module developer will need to specify a keyword, SHADOW in the INF file.

Assuming that all FD and FV images are going to be generated (based on the default value of the top-level build command), for each FV image specified the following must occur.

1. The FDF file is parsed to create a directed graph structure for each FV image, so that all leaf EFI SECTIONS are created first.

2. These leaf sections are either put into encapsulated sections or put directly into an FFS file following the implied rules (or user defined rules) defined later in this document.
3. As each FFS File is created, the file is either encapsulated into another FFS file or appended to an FV image.
4. Once all of the FFS files have been placed into an FV image file, the FV file is put into an FD file at the location specified by the FD section of the FDF file.

4.4.4 EFI PCI Expansion Option ROM Images

There are two methods for creating an option ROM image, when the FDF file is specified ane when an FDF file is not present. To build from source and no FDF file is present, if the module's INF file contains the keywords, PCI_DEVICE_ID, PCI_VENDOR_ID and PCI_CLASS_CODE, the build will terminate after creating EFI files - there will be no call to the GenFds tool. These key words alsol force the creation of an option ROM image, after the EFI files have been created, using the EfiRom program to create the EFI PCI Expansion ROM image. If an FDF is present, then the build tools will parse the FDF file looking for an [OptionRom] section, and create the option ROM based on the contents of this section. Note that the FDF specification permits adding binary images, such as the legacy option rom binary, as well as support for multiple architecture driver images to the option ROM image.

Option ROM images are always created in the output FV directory.

4.4.5 UEFI Applications

When building only UEFI applications, no FDF file is specified in the DSC file; the build would normal terminate after creating EFI files - there would be no call to the GenFds tool. Using an option on the build tool command line to specify building a UEFI Application forces the parsing stage to generate the top-level makefile with an alternate path. This path will force the creation of a UEFI application, after the EFI files have been created, using UEFI application specific arguments for the GenFds tool.

4.5 File Specifications

The EDK II Build is used to generate UEFI 2.3 and PI 1.2 compliant images. Additional reference modules may conform to Intel Framework Specifications only if there are no applicable UEFI or PI specification modules.

The EDK II Build Tools will only generate UEFI 2.3/PI 1.2 compliant images.

The EDK II Compatibility Package provides libraries and header files to permit building some* EDK Libraries and EDK Components referenced in an EDK II platform (DSC) file.

Note: ** indicates any EDK libraries or components that do not include assembly files and do not access flash memory can use the EDK II compatibility Package*

For some development activities, the EDK II Compatibility package can be used to develop and maintain original EDK platforms, components and libraries. This package also provides all of the tool source code used in the EDK. These tools are for building components and platform using the original EDK code. None of these tools is used for the EDK II build process. This EDK Compatibility package can also be used to generate files conforming to earlier releases of EFI and UEFI specifications.

This build specification does not cover the tools or build processes for EDK builds nor tools provide by the EDK II Compatibility Package.

The binary image files generated at the end of the $(MAKE) stage conform to the *UEFI Images* section of the UEFI specification. UEFI uses a subset of the PE32+ image format with a modified header signature. The PE32/PE32+ files are modified by the GenFw application. Note that this application will also modify an ELF image and generate a PE32/PE32+ image.

Each PE32/PE32+ file will have sections of the original "DOS Header" over-written, a new NT_HEADER (for the PeHeader) and possibly one Optional Header for 32-bit or 64-bit options.

4.6 File Extensions

The reference build system is designed to process files in the AutoGen stage with specific extensions for use in the $(MAKE) stage, producing files with intermediate extension names. For some "final" targets, such as UEFI applications, the intermediate extension is the "final" extension. The ImageGen makes use of the files with intermediate extensions to generate the final images.

4.6.1 AutoGen Stage File Extensions

Table 11 provides the extension and a description of files processed during the AutoGen stage of the build. The build_rule.txt file describes the processing rules for generating the Makefiles for the $(MAKE) stage.

Table 11. AutoGen Stage Input File Extensions

Extension	Description
.c, .cpp	C code files
.h	C header files
.asm	32 and 64-bit Windows assembly files
.s	32 and 64-bit GCC assembly files
.S	IPF GCC and Windows assembly files
.i	IPF Assembly include files
.vfr	Visual Forms Representation files
.uni	Unicode (UCS-2) files
.dxs	Dependency Expression files (depreciated)
.asl	C formatted ACPI code files – these files are processed independent from the C code files
.asi	ACPI Header Files
.aslc	C formatted ACPI table files - these files are processed independent from the C code files
.txt	Microcode text files
.bin	Microcode binary files
.bmp	Logo files used in the ImageGen stage
.ui	Unicode User Interface files
.ver	Unicode Version files

4.6.2 File Extensions for UEFI image files.

This section details the intermediate file extensions that generated by the $(MAKE) stage of the build process. This stage involves processing source files and generating dynamic objects which are further processed by the GenFw tool to create .efi files.

Table 12. $(MAKE) Stage Intermediate Output File Extensions

Extension	Description
.obj	Object files generated by $(MAKE) stage
.lib	Static Linked files generated by $(MAKE) stage
.dll	Dynamically Linked files generated by $(MAKE) stage
.aml	ACPI code files generated by $(MAKE) stage
.i, .iii	Trim and C Pre-Processor output files
.bin	Microcode files

Table 13. $(MAKE) Stage Output File Extensions

Extension	Description
.efi	Non UEFI Applications, DXE Drivers, DXE Runtime Drivers, DXE SAL Drivers have the Subsystem type field of the DOS/TE header set to **EFI_IMAGE_SUBSYSTEM_EFI_APPLICATION**, **EFI_IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER**, **EFI_IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER** and **EFI_IMAGE_SUBSYSTEM_SAL_RUNTIME_DRIVER** respectively. For a Security Module, the Subsystem type is set to **EFI_IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER**. For **PEI_CORE**, **DXE_CORE**, **PEIM**, **DXE_SMM_DRIVER**, **UEFI_APPLICATION**, **UEFI_DRIVER**, the Subsystem type is set to **EFI_IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER.**
.acpi	ASL or IASL compiled ACPI tables
.depex	Compiled dependency sections
.mcb	Microcode Binary files

Additional modifications to the files are permitted. Modifications that recommended are as follows:

```
TimeStructure can be modified to a given date using a data structure of
tm_mon,// months since January, [0,11]
tm_mday,// day of the month [1,31]
tm_year,// years since 1900
tm_hour,// hours since midnight [0,23]
tm_min,// minutes after the hour [0,59]
tm_sec,// seconds after the minute [0,59]
```

Subsystemfield is changed to one of the following:

```
EFI_IMAGE_SUBSYSTEM_UNKNOWN                  0
EFI_IMAGE_SUBSYSTEM_NATIVE                   1
EFI_IMAGE_SUBSYSTEM_WINDOWS_GUI              2
EFI_IMAGE_SUBSYSTEM_WINDOWS_CUI              3
EFI_IMAGE_SUBSYSTEM_OS2_CUI                  5
EFI_IMAGE_SUBSYSTEM_POSIX_CUI                7
EFI_IMAGE_SUBSYSTEM_EFI_APPLICATION         10
EFI_IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11
EFI_IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER      12
EFI_IMAGE_SUBSYSTEM_SAL_RUNTIME_DRIVER      13
```

The Machine value in the PE image file header is used to indicate the machine code type of the image. The following values are set for UEFI images:

```
EFI_IMAGE_MACHINE_IA320x014C
EFI_IMAGE_MACHINE_IA640x0200
EFI_IMAGE_MACHINE_x640x8664
EFI_IMAGE_MACHINE_EBC0x0EBC
```

4.6.3 ImageGen File Extensions

Table 14 and Table 15 describe intermediate file extensions and final file extensions in the ImageGen stage of the build for a platform. The ImageGen stage takes the output of the $(MAKE) stage (typically the .efi files) and converts the files into EFI section files using the GenSec tool. The next step combines the section files into FFS files using the GenFfs tool. Once the Ffs files have been generated, they are combined into an FV

image file using the GenFv tool. FV image files are combined into FD image files by the GenFds tool (which also controls all of the other steps in this stage).

Table 14. GenFds Image Generation: Intermediate File Extensions

Input Extension	Output Extension	Description
.efi	.pe32	EFI_SECTION_PE32
.pe32, .ui, .ver	.com	EFI_SECTION_COMPRESSION
.ui	.ui	EFI_SECTION_USER_INTERFACE
.depex	.dpx	EFI_SECTION_PEI_DEPEX or EFI_SECTION_DXE_DEPEX
.tmp, .sec	.guided	EFI_SECTION_GUID_DEFINED
.ver	.ver	EFI_SECTION_VERSION
.acpi, .aml, .bin, .bmp	.raw	EFI_SECTION_RAW
.com, .dpx, .guided, .pe32, .ui, .ver	.ffs	FFS file images
.ffs	.fv	Firmware Volume Image files
.fv	.sec	
.txt	.mcb	Microcode Binary File generated from the Microcode text files

Table 15. ImageGen Final Output File Extensions

Input Extensions	Output Extension	Description
.fv, .mcb	.fd	Firmware Device Images
.efi, .pe32	.rom	UEFI PCI OptionROM Images

For UEFI compliant PCI Option ROMs, the EfiRom tool is used to process .efi or .pe32 files into the .rom file.

For UEFI applications, the .efi file generated at the end of the $(MAKE) stage can be used directly, or, if the application will be included as part of a flash device image (all of the shell applications) the .efi file is processed using the standard steps for including a driver in an image.

4.7 Getting Started

While the EDK II User's Manual provides more detailed instructions in developing EDK II code, this section provides an overview to the steps needed to obtain code and perform a build of an emulation environment.

4.7.1 Workstation Setup

A workstation, running one of the supported operating systems should be setup on a network and the compile tools installed. An OS-specific version of a Subversion (SVN) client must be installed.

Note: *TortoiseSVN works well for Microsoft OS based workstations, while a command-line SVN client can be used on UNIX-based systems.*

Setup of the subversion client is beyond the scope of this document (refer to the EDK II website for more information on SVN).

4.7.2 Downloading Code

Using an SVN client, developers should checkout the edk2 source tree at http://edk2svn.sourceforge.net/svnroot/trunk/edk2.

Win32 binaries of the EDK II build tools are included in the BaseTools/Bin/Win32 sub-directory of the edk2 source tree. UNIX based workstations will need to build the EDK II BaseTools by using subversion to checkout the BaseTools sub-project (buildtools.tianocore.org). The SVN client is used to checkout this sub-project, located at: https://buildtools.tianocore.org/svn/buildtools/trunk/BaseTools. Unix developers should refer to the follow UNIX-specific 'Pre-build Setup' instructions below for more information.

4.7.3 Path and Target Verification

Prior to attempting the build, the developer may need to modify the configuration file, Conf/target.txt or Conf/tools_def.txt that were created by the edksetup script. The default configuration for edk2 builds uses Microsoft Visual Studio 2005 for IA32 and X64 target builds, and the Microsoft Windows DDK (WINDDK) version 3790.1830 for IPF target builds that have been installed in "standard" known locations. The default Tool Chain Tag, "MYTOOLS" uses these tools. If the tool chains are installed in non-standard locations, the Conf/tools_def.txt file should be modified to point to path where the tools are installed. If different versions of the tools (Visual Studio .NET 2003 for example) are installed, then a different tool chain tag name can be used. The TOOL_CHAIN_TAG setting in Conf/target.txt is used to control the name.

The default tools_def.txt file comes with the following Tool Chain Tag names defined: VS2003, VS2005PRO, VS2005TEAMSUITE, WINDDK3790x1830, UNIXGCC, CYGWINGCC, ICC and MYTOOLS.

4.7.4 Pre-build Setup

After obtaining the edk2 sources, the developer will need to open a Command Prompt Window or Terminal Shell. The developer should change directories to the directory containing the edk2 sources downloaded per Section 4.7.2.

The system environment variable, WORKSPACE can be set (if not set, the setup script - below - will set the WORKSPACE environment variable to the current working directory). UNIX developers may need to setup additional environment variables.

4.7.4.1 Windows

The edksetup.bat script is located at the top of the edk2 source tree should be run every time you open a new command prompt window. The edksetup.bat script needs an option, --nt32, in order to build the Nt32 emulation platform. This option is required to setup the "standard" places for include (.h) and dll files needed by the Nt32 emulation environment.

4.7.4.2 Unix

The edksetup needs to be 'sourced' under based Unix shells to allow environment variables to be modified. For the commonly used 'bash' shell this is accomplished by using a period "." character, followed by a space character, followed by the edksetup script. At the root of the Edk II tree, use the following shell command to setup the Edk II build environment:

```
. edksetup.sh
```

The bash shell also has a built-in 'source' command, so the following will also work:

```
source edksetup.sh
```

Building under Unix systems requires the BaseTools source code from the http://edk2-buildtools.svn.sourceforge.net/ BaseTools project. You might see an error from the edksetup.sh script related to not having the BaseTools source. To resolve this error, download the source code from http://edk2-buildtools.svn.sourceforge.net/. Then set the EDK_TOOLS_PATH environment variable. For example, if you wanted to download the BaseTools source to ~/src/buildtools directory, you might use the following commands:

```
cd ~/src
svn co http://edk2-buildtools.svn.sourceforge.net/viewvc/edk2-buildtools/trunk/
buildtools
export EDK_TOOLS_PATH=~/src/buildtools/BaseTools
```

Now, running the edksetup.sh script should no longer give an error. At this point you can try running the 'build' tool to see if you have successfully configured the BaseTools.

Try running this:

```
build --help
```

This command should print out the help text for the BaseTools 'build' utility.

Please note the edksetup.sh script will save/load the EDK_TOOLS_PATH setting in the Conf/BuildEnv.sh file, so it is not necessary to set the EDK_TOOLS_PATH again if a new command shell is started and edksetup.sh is 'sourced' into that shell.

The 'build' utility is written in the Python language, and therefore does not require pre-compilation before it can be used. But, there are several other 'BaseTools' utilities which must be compiled before they can run. This should be relatively easy if the machine has the standard compiler tools. In the best case, the developer should just be able to type the following command:

```
GenFv --help
```

And then run the 'make' command suggested by the output of the command.

Being able successfully run the BaseTools does not mean that your build environment is fully set up. If you want to use a GCC based compiler for UEFI/PI development, you should refer to the GCC section of this document for instructions on how to build the UEFI/PI GCC cross-compiler. It is also likely that you will need to modify the Conf/tools_def.txt and Conf/target.txt files.

4.7.5 Building Nt32 Emulation Environment

On a Windows development workstation, once these tag names and paths have been checked, the developer only needs to type "build" to build the Nt32 emulation platform. In order to run the emulation environment, type: build run after the build successfully completes.

If the developer experiences problems with the build, the most typical issues are related to the paths and tool chain tag name used by the build. The following are most common:

1. Forgetting to use the --nt32 switch on edksetup.bat. To resolve this issue:
   a Rerun: edksetup.bat --nt32.
   b Retry the build.
2. Compiler tools not found. To resolve this issue:
   a Either the wrong tool chain tag was used, or the location of the tools for the tool chain tag were not found.
   b Verify the files, Conf/target.txt and Conf/tools_def.txt.

4.7.6 Simple Build Environment Tests

There are two rudimentary build tests that you can use to test out your build environment. These two tests are building the MdePkg and the MdeModulePkg.

The MdePkg is about the most simplistic build test possible. It will only build some libraries. The MdeModulePkg will actually build some UEFI image binaries, and if it successfully completes, there is a good chance that you can successfully develop UEFI applications.

To build the MdePkg, modify the Conf/target.txt file to match your tool chain and target architecture (IA32, X64 or IPF). You can optionally set your ACTIVE_PLATFORM to MdePkg/MdePkg.dsc, but this can also be specified as a parameter to the build utility. If you set the ACTIVE_PLATFORM, then you should be able to just run the 'build' command from your shell prompt to start the BaseTools/edk2 build process.

If the MdePkg builds successfully, try the MdeModulePkg. After building the MdeModulePkg, you should be able to find some UEFI images (*.efi) underneath the $WORKSPACE/Build directory. This is a sign that your build environment is set up correctly.

5
MetaData File Specifications

This chapter defines the format of three files used by the build. The three files are: tools_def.txt, which defines the location and options for third party tools, target.txt, which defines the top level default configuration, and build_rule.txt, which specifies the rules for creating binary files.

Templates for these files are in the BaseTools/Conf directory. The edksetup script installs the active copies of these files into the WORKSPACE/Conf directory only if they do not exist.

5.1 Meta-Data File Format

The following subsections describe, at a high level, the different parts of meta-data files.

5.1.1 Section Entries (build_rule.txt only)

The build rule file consists of sections delineated by section names enclosed within square "[]" brackets. Section names are case-insensitive. The different sections and their usage are described below. The text of a given section can be used for multiple section names by separating the section names with a comma. For example:

```
[Microcode-File.USER_DEFINED, Microcode-File.Microcode]
```

Following is an example that does not use a comma:

```
[C-Code-File]
```

The content below each section heading is processed by the parsing utilities in the order that they occur in the file.

5.1.2 Comments

Within a meta-data file, comments are encouraged, with the hash "#" character identifying a comment. In line comments terminate the processing of a line. In line comments must be placed at the end of the line, and may not be placed within the section ("[", "]") tags. Comment characters can be at the start of a line, or after a data element (there must be one or more whitespace characters between the data element and the comment character. Examples:

```
# this is a comment line
[Unicode-Text-File] # This is also a valid comment.

[Unicode-Text-File # This is not valid]
```

The last example is not valid, as the section header data element format is `[text]` with the square brackets included as part of the data element.

5.1.3 Valid Entries

All entries must appear on a single line, with entries terminated by either a new line, or

a comment. Comments must be scripting style, using the hash "#" character. See previous section above for details.

5.1.4 Macro statements (tools_def.txt only)

The use of MACRO statements is limited in EDK II tools_def.txt meta-data file to be local to the meta-data file. The format and usage for the macro statements is:

```
DEFINE MACRO = Path

$(MACRO)/filename.foo
```

Any defined MACRO will be expanded by tools when they encounter the entry in the section. Three environment variables, $(WORKSPACE), $(EDK_SOURCE) and $(EDK_TOOLS_PATH) are never expanded when data is emitted to Makefiles.

The macro statements are positional, in that only statements following a macro definition are permitted - a macro cannot be used before it has been defined.

MACRO statements are permitted in DSC and FDF files to reference PATH statements, assign values to PCDs and to provide a minimum level of directive statements - refer to the corresponding specification for additional details.

5.1.5 Conditional Directive Statements

The conditional statements may appear anywhere within the file. Conditional directive sections can be nested. All conditional directives must evaluate to either “True” or “False.” Directive statements are in-fix expressions that are evaluated left to right, with items within parenthesis evaluated before the outer expressions are evaluated. Use of parenthesis is encouraged to remove ambiguity.

Prototype

```
<Conditional>          ::=  <IfStatement> <EOL>
                            [<Conditional>]{0,}
                            <Statements>{1,}
                            [<ElseIfConditional>]{0,}
                            [<Conditional>]{0,}
                            [<ElseConditional>]{0,1}
                            [<Conditional>]{0,}
                            "!endif" <EOL>

<IfStatement>          ::=  "!if" <Expression>

<Statements>           ::=  {<Sections>} {<SubSections>}
                            {<SectionStatements>}

<ElseIfConditional>    ::=  "!elseif" <Expression> <EOL>
                            <Statements>{1,}

<ElseConditional>      ::=  "!else" <EOL>
                            <Statements>{1,}

<Expression>           ::=  ["("] <VariableOrTest>
                              [<OP> <VariableOrTest>]* [")"]

<VariableOrTest>       ::=  {"$(" <Symbol> ")"} {<AritmeticTest>}

<Symbol>               ::=  (a-zA-Z)(a-zA-Z0-9_){0,}

<ArithmeticTest>       ::=  ["("] <Variable> <ArithmeticOp> <Argument>
["")"]

<Arugment>             ::=  {"0x"(a-fA-F0-9){1,}} {(0-9){1,}}

<LogicalOp>            ::=  {"AND"} {"OR"} {"XOR"} ["NOT"]

<AritmeticOp>          ::=  {"=="} {"!="} {"<"} {">"} {"<="} {">="}
```

Parameters

Statements

One or more individual elements may be within the scope of the conditional statements.

5.2 tools_def.txt

This file describes the tools used by a developer, providing the flexibility to have multiple tool chains and different profiles for each tool chain. In the simplest of terms, the file provides a variable mapping of compiler tool chains and flags. The structure of

this text file is described below.

There are three types of statements, the IDENTIFIER statement which defines a "User Interface" name for identifying this file. The second statement type is the DEFINE statement which is used to identify a fully qualified path macro, while the third type of statement is a record statement containing mappings that are processed by the build tools to generate Makefile and GNUMakefile commands that are executed by a compiler's "make" utility or function.

The left side of the record is subdivided into five groups, defined below. The build tools will process the file and assign the following priority during the parsing. After parsing the right hand <string> is substituted into the makefile using the build_rule.txt templates.

If a wildcard value is permitted, the wildcard character is the star "*" character.

For tool chains that expect to use a Windows-style nmake utility one entry, the NMAKE COMMANDTYPE is required. The *NIX-based make and gmake utilities are typically in a developer's path environment (/usr/bin). Specifying a MAKE command that will use an alternate make utility for *NIX-based tool chains is optional.

```
format: TARGET_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE = <string>
priorty:
        TARGET_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE (Highest)
        ******_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE
        TARGET_*********_ARCH_COMMANDTYPE_ATTRIBUTE
        ******_*********_ARCH_COMMANDTYPE_ATTRIBUTE
        TARGET_TOOLCHAIN_****_COMMANDTYPE_ATTRIBUTE
        ******_TOOLCHAIN_****_COMMANDTYPE_ATTRIBUTE
        TARGET_*********_****_COMMANDTYPE_ATTRIBUTE
        ******_*********_****_COMMANDTYPE_ATTRIBUTE
        TARGET_TOOLCHAIN_ARCH_**********_ATTRIBUTE
        ******_TOOLCHAIN_ARCH_**********_ATTRIBUTE
        TARGET_*********_ARCH_**********_ATTRIBUTE
        ******_*********_ARCH_**********_ATTRIBUTE
        TARGET_TOOLCHAIN_****_**********_ATTRIBUTE
        ******_TOOLCHAIN_****_**********_ATTRIBUTE
        TARGET_*********_****_**********_ATTRIBUTE
        ******_*********_****_**********_ATTRIBUTE (Lowest)
```

All entries in this file are case-sensitive.

Prototype

```
<ToolsDef>          ::=  "IDENTIFIER" [<Word>]{1,}
                         [<DefineStatements>]*
                         [<Entries>]*

<DefineStatements>  ::=  "DEFINE" <MACRO> = <ABSENVPATH> <EOL>

<MACRO>             ::=  <Word>

<ABSENVPATH>        ::=  {<AbsolutePath>} {<Environ>} [<PATH>]
```

(Continued next page)

```
<AbsolutePath>      ::=  If OS == MICROSOFT && ENV != CYGWIN:
                           <Letter> ":\" [<PATH>]
                         else:
                           "/" [<PATH>]

<PATH>              ::=  <Word> <Sep> [<Path>]*

<Environ>           ::=  "ENV(" <SysEnvironmentVariable> ")" <Sep>

<Path>              ::=  <Word> <Sep>

<Sep>               ::=  If OS == MICROSOFT && ENV != CYGWIN:
                           "\"
                         else:
                           "/"

<Entries>           ::=  <RequiredEntry>
                         [<OptionalEntry>]*


<RequiredEntry>     ::=  <Field1> "_" <Tagname> "_" <Arch> "_" <Required>

<Required>          ::=  "MAKE_PATH" "=" <EXECPATH> <Command> <EOL>

<OptionalEntry>     ::=  <Field1> "_" <Field2> "_" <Field3>

<Field1>            ::=  {<Target>} {"*"}

<Target>            ::=  {"DEBUG"} {"RELEASE"} {<UserDefinedWord>}

<Field2>            ::=  {<TagName>} {"*"}

<Tagname>           ::=  {<PreDefinedTags>} {<UserDefinedWord>}

<PreDefinedTags>    ::=  {"MYTOOLS"} {"VS2003"} {"VS2005PRO"}
                         {"VS2005TEAMSUITE"} {"WINDDK3790x1830"}
                         {"UNIXGCC"} {"CYGWINGCC"} {"ICC"}

<Field3>            ::=  <Arch> "_" <Field4> "_" <Attributes>

<Arch>              ::=  {"IA32"} {"X64"} {"IPF"} {"EBC"} {"ARM"} {"*"}

<Field4>            ::=  {<CommandCode>} {"*"}

<CommandCode>       ::=  {"APP"} {"ASL"} {"ASM"} {"ASMLINK"} {"CC"}
                         {"DLINK"} {"PCH"} {"PP"} {"SLINK"} {"TIANO"}
                         {"VFRPP"} {<UserDefinedWord>}

<Attributes>        ::=  {<ExecAttrs>} {<FlagAttr>} {<MiscAttrs>}

<ExecAttrs>         ::=  "PATH" "=" <EXECPATH> <Command> <EOL>

<MiscAttrs>         ::=  [<DllPath>} {<Family>} {<Guid>} {<UserDefined>}

<DllPath>           ::=  {"DLL"} {"DPATH"} "=" <EXECPATH> <EOL>
```

```
<EXECPATH>          ::=  {<Definition>} {<Environ>} {<AbsolutePath>}

<Command>           ::=  <Word> ["." <Ext>]

<Definition>        ::=  "DEF(" <MACRO> ")" <Sep> [<Path>]*

<Family>            ::=  "FAMILY" "=" <SupFamily> <EOL>

<SupFamily>         ::=  {"MSFT"} {"INTEL"} {"GCC"} {"ACPICA"}

<Guid>              ::=  "GUID" "=" <RegistryFormatGuid> <EOL>

<UserDefined>       ::=  <Word> "=" <UserDefinedValues> <EOL>

<FlagAttr>         ::=  "FLAG" "=" <FlagValues> <EOL>
```

Parameters

Note that no space characters are permitted on the left side of the expression (before the equal sign). All of the keywords that make up the left side of the expression must be alphanumeric only - no special characters are permitted.

Target

A keyword that uniquely identifies the build target; the first field, where fields are separated by the underscore character. Two values, "**DEBUG**" and "**RELEASE**" have been pre-defined. This keyword is used to bind command flags to individual commands.

Users may want to add other definitions, such as, PERF, SIZE or SPEED, and define their own set of **FLAGS** to use with these tags. The wildcard character, "*", is permitted after it has been defined one time for a tool chain.

TagName

A keyword that uniquely identifies a tool chain group; the second field. Wildcard characters are permitted only if a command is common to all tools that will be used by a developer. As an example, if the development team only uses IA32 Windows workstations, the ACPI compiler can be specified as **DEBUG_*_*_ASL_PATH and RELEASE_*_*_ASL_PATH**.

Arch

A keyword that uniquely identifies the tool chain target architecture; the third field. This flag is used to support the cross-compiler features, such as when a development platform is IA32 and the target platform is X64. Using this field, a single TagName can be setup to support building multiple target platform architectures with different tool chains.

For example, if a developer is using Visual Studio .NET 2003 for generating IA32 platform and uses the WINDDK version 3790.1830 for X64 or IPF platform images, a single tag. See the MYTOOLS PATH settings in the generated Conf/tools_def.txt or provided BaseTools/Conf/tools_def.template file.

The wildcard character, "*", is permitted only if the same tool is used for all target architectures.

CommandCode

A keyword that uniquely identifies a specific command; the fourth field. Several CommandCode keywords have been predefined, however users may add additional keywords, with appropriate modifications to build_rule.txt. See Table 16 below for the pre-defined keywords and functional mappings. The wildcard character, "*", is

permitted only for the FAMILY, DLL and DPATH attributes (see *Attributes* below).

Table 16. Predefined Command Codes

CommandCode	Function
APP	C compiler for applications.
ASL	ACPI Compiler for generating ACPI tables.
ASM	A Macro Assembler for assembly code in some libraries.
ASMLINK	The Linker to use for assembly code generated by the ASM tool.
CC	C compiler for PE32/PE32+/Coff images.
DLINK	The C dynamic linker.
MAKE	Required for tool chains. This identifies the utility used to process the Makefiles generated by the first phase of the build.
PCH	The compiler for generating pre-compiled headers.
PP	The C pre-processor command.
SLINK	The C static linker.
TIANO	This special keyword identifies a compression tool used to generate compression sections as well as the library needed to uncompress an image in the firmware volume.
VFRPP	The C pre-processor used to process VFR files.

Attribute

A keyword to uniquely identify a property of the command; the fifth and last field. Several pre-defined attributes have been defined: **DLL**, **DPATH**, **FAMILY**, **FLAGS**, **GUID**, **OUTPUT** and **PATH**. Use quotation marks only if the quotation marks must be included in the flag string. The following example shows the format for the required quoted string, "**C:\Program Files\Intel\EBC\Lib\EbcLib.lib**". Normally, the quotation characters are not required as everything following the equal sign to the end of the line is used for the flag.

```
*_*_EBC_DLINK_FLAGS = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /NOLOGO
```

Table 17. Predefined Attributes

Attribute	Description
DLL	The path to the Compiler required DLLs – required for some tools to get generate debug files.
DPATH	Deprecated – use DLL. The path to the Compiler required DLLs – required for some tools to get generate debug files.
FAMILY	A flag to the build command that will be used to ensure the correct commands and flags are used in the generated Makefile or GNUMakefile, as well as to use the correct options for independent tools, such as the ACPI compiler.
FLAGS	The arguments for individual CommandCode tools.
GUID	This defines the Registry Format GUID (8-4-4-4-12).
OUTPUT	This specifies an output flag for the Assembler (ASM) command.
PATH	This is the full path and executable name for a command code.

Example

```
DEFINE VSNET2003_BIN = C:\Program Files\Microsoft Visual Studio .NET
2003\Vc7\bin
DEFINE VSNET2003_DLL = C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\IDE
DEFINE ACPI_BIN      = C:\ASL

DEFINE PEGCC_BIN32   = /opt/tiano/i386-tiano-pe/i386-tiano-pe/bin
DEFINE PEGCC_BINX64  = /opt/tiano/x86_64-pc-mingw64/x86_64-pc-mingw64/bin

################################################################################
VS2003           - Microsoft Visual Studio .NET 2003
*_VS2003_*_*_FAMILY        = MSFT

###################
# IA32 definitions
###################
*_VS2003_IA32_*_DLL          = DEF(VSNET2003_DLL)

*_VS2003_IA32_CC_PATH        = DEF(VSNET2003_BIN)\cl.exe
*_VS2003_IA32_VFRPP_PATH     = DEF(VSNET2003_BIN)\cl.exe
*_VS2003_IA32_SLINK_PATH     = DEF(VSNET2003_BIN)\lib.exe
*_VS2003_IA32_DLINK_PATH     = DEF(VSNET2003_BIN)\link.exe
*_VS2003_IA32_ASMLINK_PATH   = DEF(VSNET2003_BIN)\link.exe
*_VS2003_IA32_APP_PATH       = DEF(VSNET2003_BIN)\cl.exe
*_VS2003_IA32_PP_PATH        = DEF(VSNET2003_BIN)\cl.exe
*_VS2003_IA32_ASM_PATH       = DEF(VSNET2003_BIN)\ml.exe
*_VS2003_IA32_PCH_PATH       = DEF(VSNET2003_BIN)\cl.exe
*_VS2003_IA32_NMAKE_PATH     = DEF(VSNET2003_BIN)\nmake.exe
*_VS2003_IA32_ASL_FAMILY     = ACPICA
*_VS2003_IA32_ASL_PATH       = DEF(ACPI_BIN)\iasl.exe

*_VS2003_IA32_APP_FLAGS        = /nologo /E /TC
*_VS2003_IA32_PP_FLAGS         = /nologo /E /TC /FIAutoGen.h
*_VS2003_IA32_VFRPP_FLAGS      = /nologo /E /TC /DVFRCOMPILE /FIAutoGen.h
DEBUG_VS2003_IA32_CC_FLAGS     = /nologo /W4 /WX /Gy /c /D UNICODE \
      /O1ib2 /GL /FIAutoGen.h /EHs-c- /GF /Gs8192 /Zi /Gm
```

```
RELEASE_VS2003_IA32_CC_FLAGS   = /nologo /W4 /WX /Gy /c /D UNICODE \
      /O1ib2 /GL /FIAutoGen.h /EHs-c- /GF /Gs8192
DEBUG_VS2003_IA32_PCH_FLAGS    = /nologo /W4 /WX /Gy /c /D UNICODE \
      /O1ib2 /GL /FIAutoGen.h /EHs-c- /GF /Gs8192 \
      /Fp$(DEST_DIR_OUTPUT)/AutoGen.h.gch /Yc /TC /Zi /Gm
RELEASE_VS2003_IA32_PCH_FLAGS  = /nologo /W4 /WX /Gy /c /D UNICODE \
     /O1ib2 /GL /FIAutoGen.h /EHs-c- /GF /Gs8192
     /Fp$(DEST_DIR_OUTPUT)/AutoGen.h.gch /Yc /TC
DEBUG_VS2003_IA32_ASM_FLAGS    = /nologo /W3 /WX /c /coff /Cx /Zd /W0 /Zi
RELEASE_VS2003_IA32_ASM_FLAGS  = /nologo /W3 /WX /c /coff /Cx /Zd /W0
*_VS2003_IA32_SLINK_FLAGS      = /nologo /LTCG
DEBUG_VS2003_IA32_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 \
      /OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /MACHINE:I386 /LTCG /DLL
      /ENTRY:$(ENTRYPOINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 \
      /DRIVER /DEBUG
RELEASE_VS2003_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 \
      /OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /MACHINE:I386 /LTCG /DLL \
      /ENTRY:$(ENTRYPOINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 \
      /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text

##############################################################################
#   UINIXGCC         - UINIX GCC
*_UNIXGCC_*_*_FAMILY    = GCC

###################
# Common definitions
###################
*_UNIXGCC_*_DLINK_FLAGS     = -nostdlib -O2 --gc-sections --dll \
     --export-all-symbols --entry _$(ENTRYPOINT) --file-alignment 0x20 \
     --section-alignment 0x20
*_UNIXGCC_*_ASM_FLAGS       = -c -x assembler \
     -imacros $(DEST_DIR_DEBUG)/AutoGen.h
*_UNIXGCC_*_PP_FLAGS        = -E -x assembler-with-cpp \
     -include $(DEST_DIR_DEBUG)/AutoGen.h
*_UNIXGCC_*_VFRPP_FLAGS     = -x c -E -P -DVFRCOMPILE \
     --include AutoGen.h
*_UNIXGCC_*_MAKE_PATH       = make

###################
# IA32 definitions
###################
*_UNIXGCC_IA32_CC_PATH              = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_SLINK_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)ar
*_UNIXGCC_IA32_DLINK_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)ld
*_UNIXGCC_IA32_ASM_PATH             = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_PP_PATH              = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_VFRPP_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc

*_UNIXGCC_IA32_CC_FLAGS             = -Os -fshort-wchar \
     -fno-strict-aliasing -Wall -Werror -Wno-missing-braces \
     -c -include AutoGen.h

###################
# X64 definitions
###################
```

```
*_UNIXGCC_X64_CC_PATH                = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_SLINK_PATH             = DEF(UNIXGCC_X64_PETOOLS_PREFIX)ar
*_UNIXGCC_X64_DLINK_PATH             = DEF(UNIXGCC_X64_PETOOLS_PREFIX)ld
*_UNIXGCC_X64_ASM_PATH               = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_PP_PATH                = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_VFRPP_PATH             = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc

*_UNIXGCC_X64_CC_FLAGS               = -Os -fshort-wchar \
     -fno-strict-aliasing -Wall -Werror -Wno-missing-braces \
     -Wno-address -Wno-array-bounds -c -include AutoGen.h -D_EFI_P64
```

Note: *In the above example, all entries must be on a single line. The use of the backslash "\" character is used here for readability, however it cannot be used in the file.*

5.3 target.txt File

This file is used to filter the build so "everything" does not always get built. It also provides pointers to the tools_def.txt file, and the active build_rule.txt files. All file names are relative to the system environment variable, **WORKSPACE**. No wildcard characters are permitted in this file. All entries in this file are case-sensitive.

While the values in this file filter what will be built, the TARGET, TARGET_ARCH and TOOL_CHAIN_TAG values may also be overridden on the build tool's command line.

Prototype

```
<ToolsDef>          ::=  [<Platform>]
                         [<Target>]
                         [<TargetArch>]
                         [<ToolsDef>]
                         [<ToolTagName>]
                        <ThreadEnable>
                         [<BldRuleConf>]
                         [<MoreBldRules>]*

<Platform>     ::=  "ACTIVE_PLATFORM" "=" [PlatformFile] <EOL>

<Target>       ::=  "TARGET" "=" [TargetVal]* <EOL>

<TargetArch>   ::=  "TARGET_ARCH" "=" [Arch]* <EOL>

<ToolsDef>     ::=  "TOOL_CHAIN_CONF" "=" [ToolDefs] <EOL>

<ToolTagName>  ::=  "TOOL_CHAIN_TAG" "=" [TagName]* <EOL>

<ThreadEnable> ::=  "MAX_CONCURRENT_THREAD_NUMBER" "=" [<NumThrds>] <EOL>

<NumThrds>     ::=  [1-9][(0-9)]*

<BldRuleConf>  ::=  "BUILD_RULE_CONF" "=" [BuildRules] <EOL>

<MoreBldRules> ::=  "ADD_BUILD_RULE" "=" [MoreBuildRules] <EOL>
```

Parameters

PlatformFile

Specify the WORKSPACE relative Path and Filename of the platform DSC file that will be used for the build. This line is required only if the current working directory does not contain one or more DSC files.

TargetVal

Zero or more of the following: DEBUG, RELEASE, a user defined word in the tools_def.txt file; separated by a space character. If the line is missing or no value is specified, all valid targets specified in the DSC file will attempt to be built.

Arch

What kind of architecture is for which the binary being target. One, or more, of the following, IA32, IPF, X64, EBC or ARM.

Multiple values can be specified on a single line, using space characters to separate the values. These are used during the parsing of an DSC file, restricting the build output target(s). The Build tool target ARCH is determined by a logical AND of:

target.txt: **TARGET_ARCH** values

DSC file: [D**efines] SUPPORTED_ARCHITECTURES** tag

Command-line: *-a ARCH* option

If not specified, then all valid architectures specified in the DSC file, for which tools are available, will be built.

ToolDefs

Specify the name of the filename to use for specifying the tools to use for the build. If not specified, the file: WORKSPACE/Conf/tools_def.txt will be used for the build. This file MUST be located in the WORKSPACE/Conf directory.

TagName

Specify the name(s) of the tools_def.txt TagName to use. If not specified, all applicable Tool Chain Tag name tools will be used for the build. The list uses space character separation.

Integer

The number of concurrent threads. The default, if not specified or set to zero, is 2. Recommend setting this value to one more than the number of computer cores or CPUs of the development workstation.

BuildRules

Specify the file name to use for the build rules that are followed when generating Makefiles. If not specified, the file: **WORKSPACE**/Conf/build_rule.txt will be used.

MoreBuildRules

Specify zero or more file names (separated by a space character) to use for additional build rules that are followed when generating Makefiles. The build tools will "append" these files to the end of the **BUILD_RULE_CONF** file during processing.

Example

```
ACTIVE_PLATFORM             = Nt32Pkg/Nt32Pkg.dsc
TARGET                      = DEBUG
TARGET_ARCH                 = IA32
TOOL_CHAIN_CONF             = Conf/tools_def.txt
TOOL_CHAIN_TAG              = MYTOOLS
MAX_CONCURRENT_THREAD_NUMBER = 2
BUILD_RULE_CONF             = Conf/build_rule.txt
```

5.4 build_rule.txt

This file provides templates to specify how individual modules are compiled and linked. The tools and flags specified in the tools_def.txt file (and the optional BuildOptions in the INF and/or DSC files) are used with the templates in this file to generate makefiles. Two wildcard characters are used in this file, the star "*" character and the question mark "?" character.

The star "*" character is used to indicate that the source files can be processed in parallel, while the question mark "?" character is used to indicate sequential processing of the source files an the INF file.

This file is an INI style formatted file, with square "[]" brackets used to separate different sections and angle "<>" brackets separating content within a given section.

Sections are terminated by either another section or the end-of file.

All entries in this file are case-sensitive.

The entries in the build_rule.txt files (if one or more additional build_rule format files are specified in the target.txt file) are positional. A second section heading that matches a previous section heading will cause the tools to use the rule defined in the second section rather than the rules in the section that was found first.

Prototype

```
<BuildRules>        ::=  <SectionHeading>
                         [<Inputs>]
                         [<Outputs>]
                         [<ExtraDependency>]
                         [<Commands>]

<SectionHeading>    ::=  "[<FileArg> "]" <EOL>

<FileArg>           ::=  <FileType> [<OptArgs>]

<OptArgs>           ::=  [<VerArg>] [<NextBuild>]*

<NextBuild>         ::=  "." "Build" <FileArg>

<VerArg>            ::=  "." <Version>

<FileType>          ::=  {"Acpi-Source-Language-File"}
                         {"Acpi-Table-Code-File"} {"Assembly-Code-File"}
                         {"C-Code-File"} {"C-Header-File"}
                         {"Dependency-Expression-File"}
                         {"Dynamic-Library-File"} {"Iasm-Code-File"}
                         {"Masm16-Code-File"} {"Microcode-Binary-File"}
                         {"Microcode-File"} {"Object-File"}
                         {"Static-Library-File"} {"Unicode-Text-File"}
                         {"Visual-Form-Representation-File"} {<UserDefined>}

<Version>           ::=  <Word>

<Inputs>            ::=  "<InputFile" [<InFamilyOpt>] ">" <EOL>
                         [<InputFileSpec>]*

<Outputs>           ::=  "<OutputFile" [<OutFamilyOpt>] ">" <EOL>
                         [<OutputFileSpec>]*

<ExtraDependency>   ::=  "<ExtraDependency>
                         {[<PATH>] <filename>} {<Macro>}

<Commands>          ::=  "<Command" [<CmdFamilyOpt>] ">" <EOL>
                         [<CommandSpec>]*

<InFamilyOpt>       ::=  "." <Family> ["," "InputFile" "." <Family>]*

<OutFamilyOpt>      ::=  "." <Family> ["," "OutputFile" "." <Family>]*

<CmdFamilyOpt>      ::=  "." <Family> ["," "Command" "." <Family>]*

<Family>            ::=  {"MSFT"} {"INTC"} {"GCC"} {"ACPICA"}

<InputFileSpec>     ::=  [<FileType> "="] <FileSpec> <EOL>

<FileSpec>          ::=  {<LineList>} {<CommaList>} {<SemiList>}

<LineList>          ::=  <WildCard> "." <Extension> <EOL>
```

```
<CommaList>       ::=  <WildCard> "," <Extension> ["," <CommaList>]*

<SemiList>        ::=  <WildCard> "." <Extension> [";" <SemiList>]*

<WildCard>        ::=  {"*"} {"?"}

<Extension>       ::=  (a-zA-Z) [(a-zA-Z0-9)]*

<OutputFileSpec>  ::=  <OutDirectory> "." <Extension> <EOL>

<OutDirectory>    ::=  {"$(OUTPUT_DIR)"} {"$(DEBUG_DIR)"} <Path>

<Path>            ::=  "(+)" ["${s_dir}" "(+)"] <NameVars>

<NameVars>        ::=  {"${s_base}"} {"$(MODULE_NAME)"}

<CommandSpec>     ::=  <CommandCodes> [<FileFlags>] <EOL>

<CommandCodes>    ::=  {<CmdVar>} {<Executable>}

<CmdVar>          ::=  "'$(" <Code> ")'" ["$(" <Code> "_FLAGS)"]

<Executable>      ::=  <Word> <String>

<FileFlags>       ::=  ["$(INC}"] [flag]* [flag_attr]* [<Op>] [<File>]*

<Op>              ::=  {">"} {">>"} {"> NUL 2>&1"}

<File>            ::=  filename ["." <Extension>]
```

Parameters

Code

A keyword that uniquely identifies a specific command; the fourth field of the Conf/tools_def.txt record. Several Code keywords have been predefined - refer to the CommandCode descriptions in the tools_def.txt section.

Executable

A specific command (without extension) that is in a directory that is in the user's PATH system environment variable.

flag

A flag that is specific to a command single command, such as "-o" an output flag used by some commands.

flag_attr

A required attribute for a specific flag.

filename

A reserved variable or set of variables appended using the directory separator: "(+)" character string. Table 18 lists the reserved variables that are available and their definition of how they are interpreted.

Table 18. Reserved Variable Strings

Variable String	Description
"${src}"	Source file(s) to be built (full path)
"${s_path}"	Source file directory (absolute path)
"${s_dir}"	Source file relative directory within a module. Note: ${s_dir} is always equals to "." if source file is given in absolute path.
"${s_name}"	Source file name without path.
"${s_base}"	Source file name without extension and path.
"${s_ext}"	Source file extension.
"${dst}"	Destination file(s) built from ${src} (full path)
"${d_path}"	Destination file directory (absolute path)
"${d_name}"	Destination file name without path.
"${d_base}"	Destination file name without extension and path
"${d_ext}"	Destination file extension

Table 19 lists the valid macro statements that are legal within this file.

Table 19. Valid Macro Values

Macro String	Description
$(WORKSPACE)	The current Workspace directory
$(OUTPUT_DIR)	Directory for intermediate files for building a module
$(DEBUG_DIR)	Directory for files used to debug a module
$(BUILD_DIR)	All files for building a platform will be put in this directory
$(BIN_DIR)	Common directory for executable files
$(FV_DIR)	Directory to store flash image files
$(INC)	Search path of current module
$(INC_LIST)	A file containing search pathes of current module
$(LIBS)	Static library files of current module
$(<tool>_FLAGS)	Tools flags of current module
$(MODULE_NAME)	Current module name
$(MODULE_TYPE)	Current module type
$(ARCH)	Architecture of current module
$(TOOLCHAIN)	Tool chain used to build current module
$(TARGET)	Target of current module (DEBUG/RELEASE)
$(<tool>)	Path of tool
$(EDK_TOOLS_PATH)	Path of build tools
$(INC)	Search path of the current module
$(INC_LIST)	A file containing search paths of the current module.
$(<FILE_TYPES>)	File list of each file type **Note**: The macro name is derived from file type name. For example, C-Code-File will have C_CODE_FILES macro.
$(<FILE_TYPES_LIST>)	This Macro points to a file containing a list of files for a given file type. **Note**: Note: The macro and file name are derived from file type name. For example, C-Code-File will have C_CODE_FILES_LIST macro pointing to $(OUTPUT_DIR)/ c_code_files.lst. The list file and macro name will be generated only when this macro is used in command line. This is intended to get over the long command line limitation.
$(MAKE_FILE)	This Macro points to the individual module's MAKEFILE
$(CP)	System copy command
$(MV)	System move command
$(RM)	System delete command
$(MD)	System create dir command
$(RD)	System remove dir command

Example

```
[C-Code-File]
    <InputFile>
        ?.c
        ?.C
        ?.cc
        ?.CC
        ?.cpp
        ?.Cpp
        ?.CPP

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj

    <Command.MSFT, Command.INTEL>
        "$(CC)" /Fo${dst} $(CC_FLAGS) $(INC) ${src}

    <Command.GCC>
        "$(CC)" -o ${dst} $(CC_FLAGS) $(INC) ${src}

[C-Header-File]
    <InputFile>
        *.h, *.H

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME).gch
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj

    <Command>

[Visual-Form-Representation-File]
    <InputFile>
        ?.vfr
        ?.Vfr
        ?.VFR

    <OutputFile>
        $(DEBUG_DIR)(+)${s_dir}(+)${s_base}.c

    <Command>
        "$(PP)" $(VFRPP_FLAGS) $(INC) ${src} >
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        Trim --vfr-file -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        -mkdir ${d_path} > NUL 2>&1
        VfrCompile -od ${d_path} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii
```

6
Build Environment

This chapter details the supported build environments (developer workstations) for EDK II development. It also covers the tool configuration files that describe the developer's tool environments. Note that the term "development platform(s)" referenced throughout this document means the workstation a developer using to write code and build a target binary. The architecture for the target binary does not have to match the same architecture as the developer's workstation.

Note: *Path and Filename elements within the Build Meta-Data files and command-line arguments are case-sensitive in order to support building on UNIX style operating systems.*

All builds are performed within a "WORKSPACE," which is defined as a contiguous directory structure on a development workstation. Since EDK II builds may incorporate EDK components (and their required libraries) the EDK_SOURCE directory (EDK tree structure) can reside outside of the WORKSPACE directory tree. Multiple workspaces are allowed on any given development workstation. Each workspace has its own configuration files for the build tools' meta-data. All builds occur using code from the defined WORKSPACE and/or EDK_SOURCE directories (see Environment Variables, below).

A build is always performed within the context of a "platform" defined in a single workspace. Multiple platforms can be defined in any one workspace. While some developers will not be building actual platform firmware; the platform definition file (DSC) format is suitable for OptionROM and stand-alone application development, as well as flexible enough to create binary distribution code for individual modules as well as a full platform firmware file.

While the EDK II Subversion development tree from http://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2 contains the Win32 binaries for building EDK II modules, only one set of EDK II build tools is required on a development system. The source code for all EDK II build tools is maintained on the TianoCore.org server, under the buildtools/trunk/BaseTools project. The source code for these tools is written in either generic C or Python. Developers using UNIX style operating systems for the development workstations will need to obtain the sources and build the tools separately.

6.1 Environment Variables

There are two required system level environment variables that must be set, and two optional environment variables.

6.1.1 Required Environment Variables

The first of the two required variables is WORKSPACE. This variable points to the root directory of an EDK II directory tree. Since developers will normally have more than one development tree on their workstation, this environment variable is used to identify the current working directory tree. The following two lines are an example of setting this variable, the first in a Microsoft Windows Command Prompt Window, while the

second represents setting the variable in a UNIX terminal bash shell.

```
set WORKSPACE=C:\MyWork\Proj1\edk2
export WORKSPACE=/usr/local/src/proj1/edk2
```

The second required environment variable, **EDK_TOOLS_PATH**, required points to the directory containing the Bin directory for the BaseTools directory. When using Subversion to obtain the EDK II project HEAD (most recent version of every file) a BaseTools directory, containing setup scripts, Win32 binary executables of the build tools, template files and XML Schema files is included. Only one copy of the BaseTools directories needs to be installed on a workstation (although multiple copies are permitted, such as having one in each workspace). The **EDK_TOOLS_PATH** variable must point to the directory containing the BaseTools Bin and Conf directories. The following lines are an example of setting this variable in a Microsoft Windows Command Prompt window. The first line sets an absolute path to single location, outside of the workspace, while the second line uses tools located within the workspace.

```
set EDK_TOOLS_PATH=C:\Tools
set EDK_TOOLS_PATH=%WORKSPACE%\BaseTools
```

6.1.2 Optional Environment Variables

The optional environment variables are needed to build EDK components and libraries for use in an EDK II platform. Some EDK components and libraries can be used without modifications, while other EDK components and libraries will require porting to the new EDK II development environment.

The first optional environment variable is EDK_SOURCE. This must point to either the head of an existing EDK directory tree (not the EDK II directory) or the EDK II's EdkCompatibilityPkg directory.

The second optional environment variable, EFI_SOURCE, is needed if the EDK_SOURCE environment variable is set and an EDK component and/or library is located outside of the EDK_SOURCE tree. If these values are not set, the reference build system will automatically set both values to point to the EdkCompatibilityPkg directory in the WORKSPACE.

6.2 Build Scoping

The reference build process was designed for maximum flexibility. The meta-data files and command line options enable the developer to build only what they need, rather than having to build a single platform from scratch. Multiple versions of a platform and/or module can be built, as well as just a single module within the context of a platform. This section of the document describes the techniques provided to limit what is built.

Typically, the target.txt file is used to limit the scope of a build, restricting the build to specified values for Platform, Architectures, Targets and Tool Chains. With all values in this file commented out, the build system will build all valid targets for all architectures where tools exist. However, options specified on the build tool's command line will override the TARGET, TARGET_ARCH and TOOL_CHAIN_TAG values.

6.2.1 The precedence of what (platform or module) gets built

Highest

- Content of the current working directory

If the current working directory contains an INF file, then only the module is built in the context of an ACTIVE_PLATFORM, otherwise the following apply.

- build.exe option statements (command-line options)
  - where options can be -(a-zA-Z), --(a-zA-Z)[(a-zA-Z09)]*, -(a-zA-Z) <Something> or --(a-zA-Z)[(a-zA-Z09)]* <Something>; where something is just that, can be a filename, a directory or some other thing like an assignment FOO=1.
- - target.txt file's ACTIVE_PLATFORM statement

Note: *There are two different options, the -p option specifies the ACTIVE_PLATFORM to be used for a build, so that if the current working directory contains a module INF file, then the module will be built in the context of the ACTIVE_PLATFORM. Note: if the INF file is not listed in the ACTIVE_PLATFORM's dsc file, the build will result in an error. The -m option is used to specify building an individual module (in the context of the ACTIVE_PLATFORM.*

Lowest

- - Content of the current working directory

If the ACTIVE_PLATFORM value is not set using the methods above, then, if the current working directory contains a DSC file, then the platform is built (unless a specific module is specified by the -m option on the command line.

6.2.2 The precedence of the TARGET value

It is possible to build more than one TARGET (i.e., DEBUG, RELEASE, etc.) with a single build command. The precedence of the TARGET value is:

Highest

- build.exe -b TARGET option statements

One or more of the -b TARGET options may be specified on the command line of the build tool .

- target.txt file statements

Lowest

- DSC file's BUILD_TARGETS statement

6.2.3 The precedence of the TARGET_ARCH values

Highest

- build.exe option statements
- target.txt file statements
- INF file's [Sources] section statements

If a module's INF file does not contain a [Sources] or [Sources.common] section, and does contain a [Sources.IA32] section, then the module is only valid for IA32 builds. The module will not be built for other architectures.

Lowest

- DSC file's SUPPORTED_ARCHITECTURES statement.

If an architecture set on a command line is not in the list of the DSC file's SUPPORTED_ARCHITECTURES statement, the command should fail.

6.2.4 Third Party tools using -t TOOL_CHAIN_TAG

It is possible to specify a different set of third party tools using the -t TOOL_CHAIN_TAG option to the build command. This option takes precedence over the target.txt file setting.

Highest

- build command -t TOOL_CHAIN_TAG option

Lowest

- target.txt file TOOL_CHAIN_TAG statement

6.2.5 Precedence of FLAGS values

The flags needed by third party tools can be specified on a file, module or platform basis. The default flags provided in the tools_def.txt file are for size optimization. These flags may be modified to provide better debugging capability. The precedence of the FLAGS values for third party tools follows. The reasoning behind this precedence is that flags are appended to a single line from the lowest to highest, with third party tools using the right most option. If a flag line for the Microsoft compiler contains /O1 (specified in the tools_def.txt file) and /Od (for example, from the DSC file's BuildOptions section), then the compiler only recognizes the /Od flag.

Highest

- DSC file, INF <BuildOptions> section statements
- DSC file, [BuildOptions] section statements
- INF file, [BuildOptions] section statements
- tools_def.txt file _FLAGS statements

Lowest

The following demonstrates the way tools should process the flags statements.

```
CCFLAGS = ToolsDef.CC_FLAGS + INF.BuildOptions + DSC.BuildOptions.CC_FLAGS +
          DSC.Inf.BuildOptions
```

6.2.6 Precedence of Macro Definitions

Additional build scoping can be implemented using the DSC and FDF directive statements in combination with command line options for the build tool. (Note that all command line options for the build tool are passed to the GenFds tool after the make portion of the build completes. Directive statements (defined in the DSC and FDF specifications) include `!ifdef`, `!ifndef`, `!if`, `!else`, `!elseif` and `!endif`. These control statements are used to either include or exclude lines as the parsing tool processes these files. The build and GenFds tools use the -y, --define command line options with an argument formatted: `MACRO_NAME "=" value`. If the `"=" value` is omitted, the

MACRO_NAME is assigned a value of **TRUE**.

Macro values must be defined prior to using them in directive statements or for PCD values. The following provides the rules for obtaining macro values.

Highest

- Command-line, -D flags (left most has higher priority)
- FDF file, DEFINE statements
- FDF file, DEFINE statements in the [defines] section
- DSC file, Component INF DEFINE statements embedded in <subsections>
- DSC file, DEFINE statements in sections following the [defines] sections
- DSC file, DEFINE statements in the [defines] section
- tools_def.txt file, flags

Lowest

6.2.7 Precedence of PCD Values

The values that are assigned to individual PCDs required by a build may come from different locations and different meta-data files. The following provides the rules required to assign a value to a PCD.

Highest

- Command-line, -D flags
- FDF file, SET statements within a section
- FDF file, grammar describing assignment of PCD values
- FDF file, SET statement in the [defines] section
- DSC file, Component INF <Pcd*> section statements
- DSC file, SET statements following the global [Pcd*] sections
- DSC file, global [Pcd*] sections
- DSC file, SET statements in the [defines] sections
- INF file, PCD sections, Default Values
- DEC file, PCD sections, Default Values

Lowest

6.3 Third Party Tool Flags

The tools_def.txt file provides various flags for third party tools. The following tables provide a summary of these "Best Known" options.

Note: *A reserved keyword, MDEPKG_NDEBUG, can be used for codesize reduction purposes.*

Table 20. Standard C File Compiler Options

Microsoft	Intel	GCC	Description
/nologo	/nologo		Do not display compiler version information
/c	/c	-c	Compile C files to object (.obj) files only, do not link
/WX	/WX	-Werror	Force warnings to be reported as errors.
/GS-	/GS-		Disable security checks
/W4	/W4	-Wall	Warning level 4 – print errors, warnings and remarks (or enable most warning messages)
/Gs8192			Control stack (8192 bytes) checking calls
/Gy	/Gy		Separate functions for linker.
/O1ib2	/O1		Optimize for minimum space, enable intrinsic functions, enable in-line expansion.
	/Oi		Enable Intrinsic functions
	/Ob2	-default-inline	Inline any function, at the compiler's discretion (same as /Qip)
		-O	Optimize output file
/GL			Enable link-time code generation
/EHs-c-			Combine /EHs- and /EHc-
	/EHs-		Disable C++ EH (no SHE exceptions)
	/EHc-		Disable extern C defaults to no throw
/GF	GF		Enable read-only string pooling
/GR-			Disable C++ RTTI
EDK II Specific Flags			
/D UNICODE	/D UNICODE	-DUNICODE	define macro UNICODE
/FIAutoGen.h	/FIAutoGen.h	--include AutoGen.h	Always include AutoGen.h file
Debug Specific Flags			
/Zi	/Zi	-g	Enable debugging information
/Gm	/Gm		Enable minimum rebuild
		-fshort-wchar	Force the underlying type for "wchar_t" to be "unsigned short"
IPF Specific Flags			
/Ox			Maximum Optimization (/Ogityb2 /Gs)
/X			ignore standard places
/QIPF_fr32			Do not use upper 96 Floating Point Registers
/Zx			Generates debug-able optimized code. Only available in the IPF cross compiler or IPF native compiler.

Table 21. Assembly Flags

Microsoft	GCC	Description
/nologo		Do not display assembler version information
/c	-c	Generate object (.obj) files, do not link
/WX		Treat warnings as errors
/W3		Warning level 3
/Cx		Preserve case in publics and externs
/coff		Generate COFF format object files
/Zd		Add line number debug info
/Zi		Add symbolic debug info (DEBUG target)
	-x assembler	Input files are in assembly language
	-imacros AutoGen.h	Accept definition of macros in AutoGen.h

Table 22. C Compiler's Preprocessor Options

Microsoft	Intel	GCC	Description
/nologo	/nologo		Do not display compiler version information
/E	/E	-E	Preprocess only; do not compile, assemble or link
/TC	/TC	-x assembler-with-cpp	Compile as .c files
/FIAutoGen.h	/FIAutoGen.h	--include AutoGen.h	Always include AutoGen.h file

Table 23. C Compiler's Preprocessor Options for VFR files ONLY

Microsoft	Intel	GCC	Description
/nologo	/nologo		Do not display compiler version information
/E	/E	-E	Preprocess only; do not compile, assemble or link
/TC	/TC	-x c	Compile as .c files
/D VFRCOMPILE	/D VFRCOMPILE	-DVFRCOMPILE	Used only for Preprocessing VFR files
		-P	Used only for Preprocessing VFR files - do not generate #line directives
/FI$(MOD_NAME)StrDefs.h			Force include of the module's StrDefs.h file.

Table 24. Pre-compiled Header (PCH) Creation Flags

Microsoft	Intel	GCC	Description
/nologo	/nologo		Do not display compiler version information
/c	/c	-c	Compile C files to object (.obj) files only, do not link
/W4	/W4	-Wall	Warning level 4 – print errors, warnings and remarks (or enable most warning messages)

/WX	/WX	-Werror	Force warnings to be reported as errors.
/Gy	/Gy		Separate functions for linker.
/GS-	/GS-		Disable security checks
/O1	/O1		Optimize for Maximum Speed
/Oi	/Oi		Enable Intrinsic functions
/Ob2	/Ob2	-default-inline	Inline any function, at the compiler's discretion (same as /Qip)
/GL			Enable link-time code generation
/EHs-	/EHs-		Disable C++ EH (no SHE exceptions)
/EHc-	/EHc-		Disable extern C defaults to no throw
/GF	/GF		Enable read-only string pooling
/Gs8192	/Gs8192		Control stack (8192 bytes) checking calls
/TC	/TC		Compile as .c files
/Yc			Create the .pch file
/Gm			Enable minimal rebuilds
/FpAutoGen.h.gch			
/X	/X		Ignore standard places
/Zi	/Zi		Produce debugging information

Table 25. Static Linker Flags

Microsoft	GCC	Description
/nologo		Do not display compiler version information
/LTCG		Use link-time code generation

Table 26. Dynamic Linker Flags

Microsoft	GCC	Description
/NOLOGO		Do not display compiler version information
/NODEFAULTLIB	-nostdlib	Disable using default libraries
/IGNORE:4086	N/A	USE /Gz option instead
/OPT:ICF=10		Perform identical COMDAT folding (10 iterations) to remove duplicates.
/MAP	-Map filename.map	Create a map file.
/ALIGN:32	--section-alignment 0x20 --file-alignment 0x20	Use 32-byte alignment instead of the default 4K
/MACHINE:$$	N/A	Where $$ is one of: I386, AMD64 or IA64
/DLL	--dll	The output is a DLL
/LTCG		Use link-time code generation
/ENTRY:$(ENTRYPOINT)	--entry _$(ENTRYPOINT)	The function that specifies a starting address.
/SUBSYSTEM:CONSOLE	--subsystem console	Do not use the EFI_* subsystem interface, as this is EFI 1.0 compliant, not UEFI 2.1.
/SAFESEH:NO		Do not produce an image with a table of safe exception handles
/BASE:0	--image-base 0x0	Base address is always 0, and will be adjusted later by the build tools when creating images.
/DRIVER		Specify Kernel mode
/DEBUG		Create debugging information
	-O2	Optimize
	--gc-sections	Enable garbage collection of unused input sections
	--export-all-symbols	All global symbols in the objects used to build a DLL will be exported by the DLL.

7
AutoGen Stage

7.1 Overview

This chapter describes in detail the steps that are accomplished by the AutoGen stage, which is the first step of building a platform or a module.

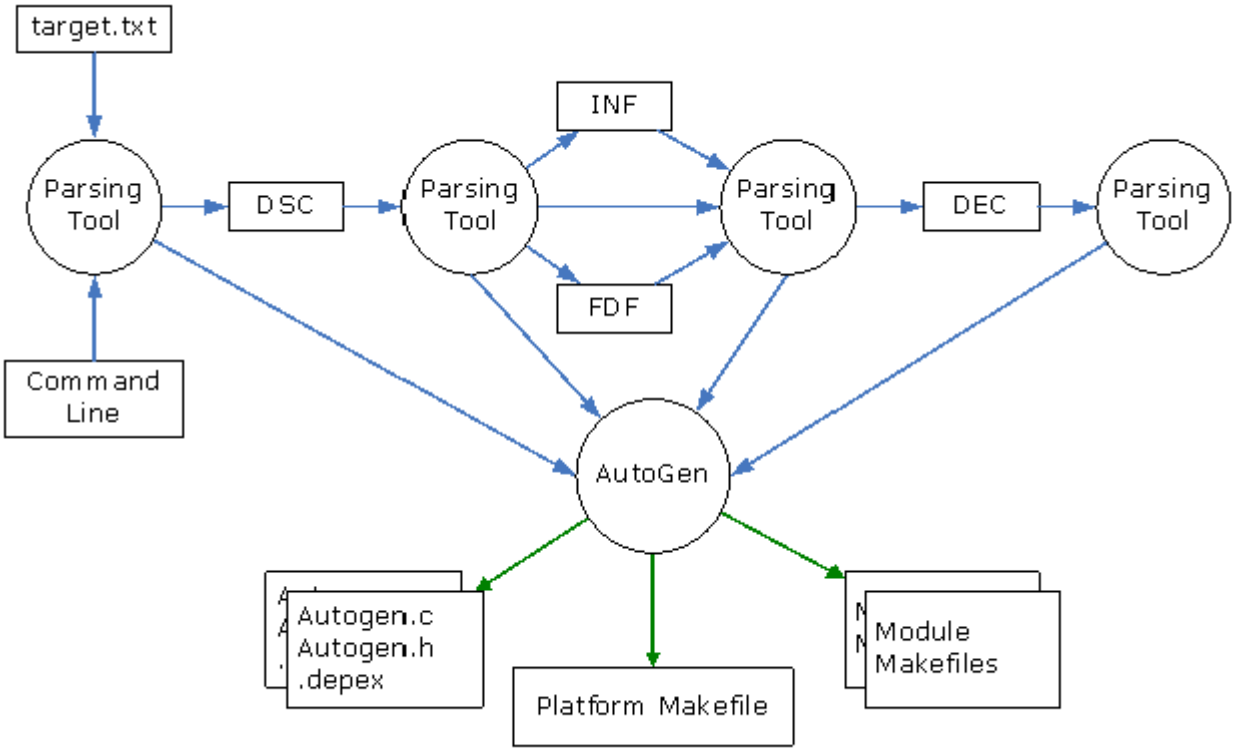


Figure 19. EDK II AutoGen Process

The first file the build tool is looking for in AutoGen stage is target.txt in directory $(WORKSPACE)/Conf. All the configurations in target.txt can be overridden by command line options of build tool. If no platform description file is specified in either target.txt and command line, the build tool will try to find one in current directory. And if build tool finds a description file of a module (INF file) in current directory, it will try to build just that module only rather than building a whole platform.

Once the build tool gets what to build and how to build, it starts to parse the platform description file (DSC). From the DSC file, the build tools will locate the INF files for all modules and libraries, as well as other settings of the platform (including DEC specified default values for PCDs used by modules and libraries that do not have values specified in the DSC file).

From module description files, the build tool will find out what package description files

the module depends on. In this way, the build tool will find out and parse all modules and packages that make up a platform.

The next thing to do in the AutoGen stage is to generate files required to build a module. The files include: AutoGen.h, AutoGen.c, .depex and makefile.

AutoGen.c and .depex files will not be generated for library modules, and .depex file is generated only if there's [depex] section found in the module's INF file.

Each module found in DSC file will have a makefile generated for it. A top-level makefile will be generated for the platform, from which all modules' makefile will be called.

Note: *When building a module, only the module's makefile will be called.*

7.2 Auto-generation Process

This section covers, in sequence, the steps taken by the build.exe tool.

7.2.1 Determine What to Build

The build tool will use following algorithm to determine what will be built.

```
If ("-t <DscFile>") {
  // Command line option specified

  ActivePlatform = <DscFile>;

} ElseIf (<ACTIVE_PLATFORM> specified in $(WORKSPACE)/Conf/target.txt) {

  ActivePlatform = <ACTIVE_PLATFORM>;

} ElseIf (one <DscFile> found in current working directory) {

  ActivePlatform = <DscFile>;

} Else {
  if (Number of DscFiles > 1) {

    PrintError("There are %s DSC files in the folder. Use '-p' to specify one.",
NumDscFiles);

  } else {

    PrintError("No active platform specified in target.txt or command line!\n
Nothing to build.");

  }

  BreakTheBuild();

}

If (("-m <InfFile>") || (one <InfFile> found in working directory)) {
  // Either a command line option was specified, or one and only
  // one INF file was found in the current working directory.

  ActiveModule = <InfFile>;
  BuildMode = "SingleModuleBuild";

} Else {

  ActiveModule = NONE;
  BuildMode = "PlatformBuild";

}

Parse( $(WORKSPACE)/Conf/target.txt );
Parse( ActivePlatform );

If ("-a <ArchListFromCommandLine>" ) {
  // command line option given

  ActiveArchList = Intersection(
    <ArchListFromCommandLine>,
    <ArchListFrom(ActivePlatform)>
  );
```

```
} Else {

  ActiveArchList = Intersection(
    <ArchListFromTarget.Txt>,
    <ArchListFrom(ActivePlatform)>
  );

}

If (ActiveArchList == NULL) {
  if (ArchListFromCommandLine != NULL) {
    PrintError("The architecture(s) specified on the command line (%s) are not
valid for the active platform (%s\n",
     ArchListFromCommandLine,
     ArchListFrom(ActivePlatform) );
  } else {
    PrintError("The active platform cannot be built, the architectures (%s) are
not supported.\n", ArchListFrom(ActivePlatform));
  }
  BreakTheBuild();

}


If ("-a <TargetListFromCommandLine>") {
  // command line option given

  ActiveTargetList = Intersection(
    <TargetListFromCommandLine>,
    <TargetListFrom(ActivePlatform)>
  );

} Else {

  ActiveTargetList = Intersection(
    <TargetListFromTarget.Txt>,
    <TargetListFrom(ActivePlatform)>
  );

}

If (ActiveTargetList == NULL) {
  if (TargetListFromCommandLine != NULL) {
    PrintError("Target (%s) specified on the command line is not valid for this
platform (%s).\n", TargetListFromCommandLine, TargetListFrom(ActivePlatform) );
  } else {
    PrintError( "Target (%s) is not specified in the target.txt file.\n",
TargetListFrom(ActivePlatform) );
  }
  BreakTheBuild();

}


If ("-t <ToolChainTag>") {
```

```
  // command line option given

  ActiveToolChain = <ToolChainTag>

} ElseIf (<TOOL_CHAIN_TAG> specified in $(WORKSPACE)/Conf/target.txt) {

  ActiveToolChain = <TOOL_CHAIN_TAG>

} Else {
  if (ToolChainTag != NULL) {
    PrintError( "Tool chain specified on the command line (%s) is not specified
in the tools_def.txt file.\n", ToolChainTag);
  } else {
    PrintError( "Tool chain specified in target.txt (%s) is not specified in the
tools_def.txt file.\n", TOOL_CHAIN_TAG);
  }
  BreakTheBuild();

}

Build( ActivePlatform, ActiveModule, ActiveArchList, ActiveTargetList,
ActiveToolChain, BuildMode );
```

7.2.2 Parse file pointed to by TOOL_CHAIN_CONF

The file specified by TOOL_CHAIN_CONF (in target.txt) is the tool chain definition file (toold_def.txt) that contains all the definitions of external tools used to build modules and platforms, in the form of "name=value". The definition of a tool includes the path of the executable, the path of dynamic libraries the executable needs, and command line options. Each set of tools can be referenced by a tag name either in the command line or in target.txt. For example, WINDDK3790x1830 is used to refer a set of tools from WINDDK of version 3790x1830.

The parser of the tool chain definition file needs to expand macros and "*" in the tool definitions. The expanded definitions are put in a database for easier access later. For example, if one overrides a tool's options in DSC or INF file, the tool will look up the tool's definition in the database and append the options to the end of options in the file specified by TOOL_CHAIN_CONF.

Note: *The supported third party compiler tools will use the right most (or last) option it encounters, permitting appended options to override options specified first. For example, specifying a compiler option (FLAG) line: /Od /c /Og will result the compiler only processing /c /Og, ignoring the /Od flag.*

The final result after AutoGen stage is that macros named by <TOOLCODE> and <TOOLCODE>_FLAGS> will be generated in module's makefile. For example, "CC" and "CC_FLAGS" macros will be generated in the makefile for the compiler tool. The path of dynamic libraries will be prefixed to system's PATH environment by the build tools, so that the tools used in makefile can be called correctly.

7.2.3 Parse build_rule.txt specified by BUILD_RULE_CONF and ADD_BUILD_RULE

The files specified by BUILD_RULE_CONF and/or ADD_BUILD_RULE (in target.txt) contain command steps used to build the source files into intermediate files and then

intermediate files into final image files to be put into FV/FD. The type of source files and intermediate files are determined by the file extension. That means the same extension cannot be used to represent different file types. But one type of file can have more than one file extension. A single file can only have a single extension.

The parser of this file will convert the contents of the file into a build rule database. Each item in this database will have tool chain family, input file information, output file information and command information. Whenever a source file is found in module's INF file, the build tools will attempt to find a build rule in the database corresponding to the input file's extension, and then use the output file as input file information to find another build rule, until no build rule uses the output file information as its input file. If there's no build rule for a type of source file, the build tools just skip it. But if there's build rule for it, one or more makefile targets will be generated for it.

The sequence of build rules applied to source files and intermediate files determines the dependency relationship between targets in makefile. One type of file cannot be used in more than one build rule as an input file and the build rules must not be cyclic.

7.2.4 Parse DSC, FDF, INF, DEC files

From the DSC file, the build tools collect the mapping between library classes and library instances, PCD data for the whole platform, the list of INF file of modules specified for the platform, and the build output directory. Optionally, the name of the flash image layout description (FDF) file and build options specific to the platform are also obtained.

The build tools try to parse the INF file one by one, including the INF file for library instances. From the INF file, the build tools collect information such as source file list, library class list, package list, GUID/Protocol/PPI list, PCD list, etc.

After all INF files are parsed, the build tools retrieve the list of all of the dependent DEC files and then parse them. From the DEC file, the build tools will get the information such as common include folders, the values of GUID/Protocol/PPI, the default setting of all PCDs in the package, etc.

Finally, the build tools attempt to parse FDF file, if specified, obtained from DSC file. Parsing FDF file at this time is just for the PCD information which might be used by some modules, and merge these PDC values into the information set of PCDs in DSC file.

7.2.5 Post processing

Once all files are parsed, the build tools will do following work for each EDK II module:

- Resolve the library classes to library instances, inherit and resolve library classes from them recursively, until no new library instances are found.
- Re-order the library instances according to the consuming relationship and their constructors.
- Inherit GUIDs, Protocols, PPIs, and PCDs from all library instances obtained above, and determine values or type of them. The value of a GUID, Protocol or PPI is defined in DEC file. The value and type of a PCD are obtained from DSC file, or DEC file if it cannot be found in DSC file.
- Inherit library instance dependency ([depex] sections) expressions if a module does not list a separate dependency file.

7.3 Auto-generated code

The section covers, in sequence, the processes used to generate code files that will be used during the build.

7.3.1 Dependency expression file

The dependency expression file (.depex) is generated from the [depex] section in module's INF file, if the section presents, or .dxs file if DPX_SOURCE definition is found in INF file. The GUID used in [depex] section must be the GUID C name.

First, the GUID C name in the dependency expression string will be converted into its value in C structure format. Then the expression string will be converted into postfix notation. Before saving to a file, the operator and GUID value in the postfix notation will be converted to their binary value.

For example, the following dependency expression

```
NOT (gEfiHiiDatabaseProtocolGuid AND gEfiHiiStringProtocolGuid) OR
gPcdProtocolGuid
```

will be converted to

```
include_statement($(MODULE_BUILD_DIR)/OUTPUT/$(BASE_NAME).dxs, "
  // PUSH
  02
  // gEfiHiiDatabaseProtocolGuid
  72 c1 9f ef b2 a1 93 46 b3 27 6d 32 fc 41 60 42
  // PUSH
  02
  // gEfiHiiStringProtocolGuid
  74 69 d9 0f aa 23 dc 4c b9 cb 98 d1 77 50 32 2a
  // AND
  03
  // NOT
  05
  // PUSH
  02
  // gPcdProtocolGuid
  06 40 b3 11 5b d8 0a 4d a2 90 d5 a5 71 31 0e f7
  // OR
  04
  // END
  08
");
```

The binary dependency expression file will be generated in $(MODULE_BUILD_DIR)/OUTPUT with .depex file extension.

7.3.2 HII String Pack

The human-readable HII string pack data is stored in Unicode format in .uni files. The build tools will do following steps to convert the strings information into HII string pack data structure.

- The build tools will get all the string IDs, the associated string and language code from the .uni files. Note that the DSC file or options on the command-line may be used to filter the languages used for generating the autogen code. The RFC_LANGUAGES is a semi-colon separated, doubled quoted string of RFC 4646

language codes, while the ISO_LANGUAGES is a non-separated double quoted string of three character ISO 639-2 language codes.

- Search all source files in the include path of the module to find out which string IDs are used.
- Macros will be generated in AutoGen.h for the string IDs used. Those string IDs not used will be generated but commented out. They is just for debug purposes. For example:

```
include_statement(AutoGen.h, "
  //
  //Unicode String ID
  //
  // #define $LANGUAGE_NAME 0x0000 // not referenced
  // #define $PRINTABLE_LANGUAGE_NAME 0x0001 //not referenced
  #define STR_BOOT_FAILED 0x0002
  #define STR_BOOT_SUCCEEDED 0x0003
  #define STR_PERFORM_MEM_TEST 0x0004
  // #define STR_INTERNAL_EFI_SHELL 0x009E // not referenced
  // #define STR_LEGACY_BOOT_A 0x009F // not referenced
  // #define STR_PROCESSED_ALL_BOOT_OPTIONS 0x00A0 // not referenced
");
```

- The HII string package data will be generated in AutoGen.c in the form of a data array, with array name <ModuleBaseName>Strings. For example,

```
include_statement(AutoGen.c, "
  //
  //Unicode String Pack Definition
  //
  unsigned char PlatformBdsDxeStrings[] = {
  // Start of string definitions for fra
    0x20, 0x1A, 0x00, 0x00, 0x02, 0x00, 0x8E, 0x02,
    0x00, 0x00, 0x96, 0x02, 0x00, 0x00, 0x9E, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    // offset 0x16
    0x8E, 0x02, 0x00, 0x00,  // offset to string $LANGUAGE_NAME (0x0000)
    0x96, 0x02, 0x00, 0x00,  // offset to string
                             //$PRINTABLE_LANGUAGE_NAME (0x0001)
    …
    …
    // string $LANGUAGE_NAME offset 0x0000028E
    0x66, 0x00, 0x72, 0x00, 0x61, 0x00, 0x00, 0x00,
    // string $PRINTABLE_LANGUAGE_NAME offset 0x00000296
    0x46, 0x00, 0x72, 0x00, 0x61, 0x00, 0x6E, 0x00,
    0xE7, 0x00, 0x61, 0x00, 0x69, 0x00, 0x73, 0x00,
    0x00, 0x00,
    …
    …
    // strings terminator pack
    0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  };
");
```

7.3.3 AutoGen.h file

The code generated in AutoGen.h includes:

- Prototypes of constructor and destructor from the library instances the module will link against
- Prototypes of entry and unload image entry points
- Global variable definitions for GUID/Protocol/PPI values used
- Global variable definitions and the database of PCDs used
- Unicode string database definitions.

The file will contain:

7.3.3.1 Header prologue

The macro name is composed with GUID value of INF file.

```
include_statement(AutoGen.h, "
  #ifndef _AUTOGENH_6987936E_ED34_44db_AE97_1FA5E4ED2116
  #define _AUTOGENH_6987936E_ED34_44db_AE97_1FA5E4ED2116
");
```

7.3.3.2 Global macro definitions

If they are defined in INF file. Un-defining them first is for backward compatibility with R8 module build, because these macros are not defined in INF file of R8 modules but passed via compiler option.

```
include_statement(AutoGen.h, "
  #undef EFI_SPECIFICATION_VERSION
  #define EFI_SPECIFICATION_VERSION 0x00020000

  #undef EDK_RELEASE_VERSION
  #define EDK_RELEASE_VERSION 0x00020000
");
```

7.3.3.3 Header file inclusion.

Only one header file is included.

```
include_statement(AutoGen.h, "
  #include <Base.h>
");
```

7.3.3.4 Caller ID GUID definition.

The GUID value is the same as INF file GUID. The macro"`EFI_CALLER_ID_GUID` is generated only for non-library module.

```
include_statement(AutoGen.h, "
  extern GUID  gEfiCallerIdGuid;
  // following definition is not needed for library module
  #define EFI_CALLER_ID_GUID \
    { 0x6987936E, 0xED34, 0x44db, { 0xAE, 0x97, 0x1F, 0xA5, 0xE4, 0xED,
      0x21, 0x16 } }
");
```

7.3.3.5 PCD definitions

There are differences in the generated code for library and non-library modules, which are illustrated in pseudo-code below.

7.3.3.5.1 Non-library Module

```
include_statement(AutoGen.h, "
  #define _PCD_TOKEN_<TokenCName>  <TokenNumber>
");

If ((PCD_type == FIXED_AT_BUILD) || (PCD_type == FEATURE_FLAG)) {

  If ((DatumType == 'VOID*') && (PcdValue == array)) {
    include_statement(AutoGen.h, "
     "#define _PCD_PATCHABLE_<TokenCName>_SIZE <MaxDatumSize>"

    );
  }
  include_statement(AutoGen.h, "
  #define _PCD_VALUE_<TokenCName>  <PcdValue>
  extern const <DatumType> _gPcd_FixedAtBuild_<TokenCName>;
  #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
                          _gPcd_FixedAtBuild_<TokenCName>

  ");
}
If (PCD_type == PATCHABLE_IN_MODULE) {

  If ((DatumType == 'VOID*') && (PcdValue == array)) {
    include_statement(AutoGen.h, "
      #define _PCD_PATCHABLE_<TokenCName>_SIZE <MaxDatumSize>

    ");

  }
  include_statement(AutoGen.h, "
    #define _PCD_VALUE_<TokenCName>  <PcdValue>
    extern <DatumType> _gPcd_BinaryPatch_<TokenCName>;
    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
                                    _gPcd_BinaryPatch_<TokenCName>

  ");

  If ((DatumType == 'VOID*') && (PcdValue == array)) {
     include_statement(AutoGen.h, "
       #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(SizeOfBuffer,\
         Buffer) \
         LibPatchPcdSetPtr(_gPcd_BinaryPatch_<TokenCName>, \
           (UINTN)_PCD_PATCHABLE_<TokenCName>_SIZE, \
           (SizeOfBuffer), \
           (Buffer) \
         )

    ");
  } Else {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
        (_gPcd_BinaryPatch_<TokenCName> = (Value))

    ");
```

```
  }

}

If (PCD_type == DYNAMIC) {
  include_statement(AutoGen.h, "
    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
       LibPcdGet<DatumSize>(_PCD_TOKEN_<PcdTokenCName>)

  ");
  If (DatumType == 'VOID*') {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(SizeOfBuffer,
        Buffer) \
        LibPcdSet<DatumSize>( \
          _gPcd_BinaryPatch_<TokenCName>, \
          SizeOfBuffer), \
         (Buffer) \
        )

    ");
  } Else {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
        LibPcdSet<DatumSize>(_gPcd_BinaryPatch_<TokenCName>, (Value))

    ");
  }

}

If (PCD_type == DYNAMIC_EX) {
  include_statement(AutoGen.h, "
    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
      LibPcdGetEx<DatumSize>(&<TokenSpaceGuidCName>, \
        _PCD_TOKEN_<PcdTokenCName> \
      )

  ");
  If (DatumType == 'VOID*') {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>( \
        SizeOfBuffer, Buffer) \
        LibPcdSetEx<DatumSize>(&<TokenSpaceGuidCName>, \
                        _gPcd_BinaryPatch_<TokenCName>, \
                         (SizeOfBuffer), \
                         (Buffer) \
                        )

    ");
  } Else {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
        LibPcdSetEx<DatumSize>(&<TokenSpaceGuidCName>, \
                            _gPcd_BinaryPatch_<TokenCName>, \
```

```
                                    (Value) \
                                    )

      ");
    }

  }
```

7.3.3.5.2 Library Module

```
include_statement(AutoGen.h, "
  #define _PCD_TOKEN_<TokenCName>  <TokenNumber>

");

If ((PCD_type == FIXED_AT_BUILD) || (PCD_type == FEATURE_FLAG)) {
  include_statement(AutoGen.h, "
    extern const <DatumType> _gPcd_FixedAtBuild_<TokenCName>;

    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
      _gPcd_FixedAtBuild_<TokenCName>

  ");
}

If (PCD_type == PATCHABLE_IN_MODULE) {
  include_statement(AutoGen.h, "
    extern <DatumType> _gPcd_BinaryPatch_<TokenCName>;

    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
      _gPcd_BinaryPatch_<TokenCName>
    #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
      (_gPcd_BinaryPatch_<TokenCName> = (Value))

  ");
}

If (PCD_type == DYNAMIC) {
  include_statement(AutoGen.h, "
    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
      LibPcdGet<DatumSize>(_PCD_TOKEN_<PcdTokenCName>)

  ");
  If (DatumType == 'VOID*') {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>( \
        SizeOfBuffer, Buffer) \
        LibPcdSet<DatumSize>(_gPcd_BinaryPatch_<TokenCName>, \
                                          (SizeOfBuffer), \
                                          (Buffer) \
                                          )
    ");
  } Else {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
        LibPcdSet<DatumSize>(_gPcd_BinaryPatch_<TokenCName>, (Value))

    ");
  }

}

If (PCD_type == DYNAMIC_EX) {
  include_statement(AutoGen.h, "
```

```
    #define _PCD_GET_MODE_<DatumSize>_<TokenCName> \
      LibPcdGetEx<DatumSize>( &<TokenSpaceGuidCName>, \
        _PCD_TOKEN_<PcdTokenCName> )

  ");
  If (DatumType == 'VOID*') {
    include_statement(AutoGen.h, "
     #define _PCD_SET_MODE_<DatumSize>_<TokenCName>( \
       SizeOfBuffer, Buffer ) \
       LibPcdSetEx<DatumSize>(&<TokenSpaceGuidCName>, \
         _gPcd_BinaryPatch_<TokenCName>, \
         (SizeOfBuffer), \
         (Buffer) \
       )

    ");
  } Else {
    include_statement(AutoGen.h, "
      #define _PCD_SET_MODE_<DatumSize>_<TokenCName>(Value) \
        LibPcdSetEx<DatumSize>(&<TokenSpaceGuidCName>, \
          _gPcd_BinaryPatch_<TokenCName>, \
          (Value) \
          )

    ");
  }

}
```

7.3.3.5.3 Hll string pack definitions,

These are generated only if .uni files are found. For details, please refer to section 7.3.2.

```
include_statement(AutoGen.h, "
  //
  //Unicode String ID
  //
  // #define $LANGUAGE_NAME 0x0000 // not referenced
  // #define $PRINTABLE_LANGUAGE_NAME 0x0001 // not referenced
  #define STR_MISC_BASE_BOARD_MANUFACTURER  0x0002
  #define STR_MISC_BASE_BOARD_PRODUCT_NAME  0x0003
  #define STR_MISC_BASE_BOARD_VERSION       0x0004
  // …
  // …
  // …
  extern unsigned char MiscSubclassStrings[];

  #define STRING_ARRAY_NAME MiscSubclassStrings

");
```

7.3.4 AutoGen.c file

The code generated in AutoGen.c includes:

- Calling of constructor and destructor of library instances against which the module will link
- The module load and unload entry points
- Global variables for GUID/Protocol/PPIs value used, global variables and database for PCDs used
- Unicode string pack definition.

AutoGen.c file is only generated for EDK II non-library modules. The following sections identify what lines of information are included in the file as well as pseudo-code to references on to how a variable (<var_name>) might be generated.

The file will contain:

7.3.4.1 Header files inclusion.

Which files are included is determined by module type.

```
Switch MODULE_TYPE {
  case "BASE":
  case "USER_DEFINED":
    include_statement(AutoGen.c, "
      #include <Base.h>

    );
    break;

  case "SEC":
  case "PEI_CORE":
  case"PEIM":
    include_statement(AutoGen.c, "
      #include <PiPei.h>
      #include <Library/DebugLib.h>

    ");
    break;

  case "DXE_CORE":
    include_statement(AutoGen.c, "
      #include <PiDxe.h>
      #include <Library/DebugLib.h>

    ");
    break;

  case "DXE_DRIVER":
  case "DXE_SMM_DRIVER":
  case "DXE_RUNTIME_DRIVER":
  case "DXE_SAL_DRIVER":
  case "UEFI_DRIVER":
  case "UEFI_APPLICATION"
    include_statement(AutoGen.c, "
      #include <PiDxe.h>
      #include <Library/BaseLib.h>
      #include <Library/DebugLib.h>
      #include <Library/UefiBootServicesTableLib.h>
```

```
    ");
    break;

  default:
    PrintError( "%s\n", message);
    BreakTheBuild();
}
```

7.3.4.2 Caller ID GUID variable definition.

Because not all GUID variables are required, a link-time optimization removes items that are not referenced by other parts of the code to save on space in the image.

```
include_statement(AutoGen.c, "
  GLOBAL_REMOVE_IF_UNREFERENCED GUID gEfiCallerIdGuid = {0x4A9B9DB8,
    0xEC62, 0x4A92, {0x81, 0x8F, 0x8A, 0xA0, 0x24, 0x6D, 0x24, 0x6E}};

");
```

7.3.4.3 Library Constructor Statements

If there are CONSTRUCTORs defined in [Defines] section in INF file of the library instances that are being linked to.

```
If (CONSTRUCTOR defined in INF) {

  If (MODULE_TYPE == "BASE") {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <CONSTRUCTOR> (
        VOID
      );

    ");

  }

  If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <CONSTRUCTOR> (
        IN EFI_PEI_FILE_HANDLE       FileHandle,
        IN EFI_PEI_SERVICES          **PeiServices
      );

    ");

  If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
    (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
    (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
    (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
    (MOODULE_TYPE =='UEFI_DRIVER') ||
    (MODULE_TYPE == 'UEFI_APPLICATION')) {
```

```
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <CONSTRUCTOR> (
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
      );

    ");
  }

} // End CONSTRUCTOR defined in INF

If (MODULE_TYPE == "BASE") {
  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryConstructorList (
      VOID
    )

  ");
}

If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryConstructorList (
      IN EFI_PEI_FILE_HANDLE       FileHandle,
      IN EFI_PEI_SERVICES          **PeiServices
    )

  ");
}

If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
    (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
    (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
    (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
    (MOODULE_TYPE =='UEFI_DRIVER') ||
    (MODULE_TYPE == 'UEFI_APPLICATION')) {

  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryConstructorList (
      IN EFI_PEI_FILE_HANDLE       ImageHandle,
      IN EFI_PEI_SERVICES          **SystemTable
    )

  ");
}
```

```
include_statement(AutoGen.c, "
  {

");

If (CONSTRUCTOR defined in INF) {
  If (MODULE_TYPE == "BASE") {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <CONSTRUCTOR>();
      ASSERT_EFI_ERROR (Status);

    ");
  }
  If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <CONSTRUCTOR>(FileHandle, PeiServices);
      ASSERT_EFI_ERROR (Status);

    ");
  }

  If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
      (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
      (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
      (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
      (MOODULE_TYPE =='UEFI_DRIVER') ||
      (MODULE_TYPE == 'UEFI_APPLICATION')) {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <CONSTRUCTOR>(ImageHandle, SystemTable);
      ASSERT_EFI_ERROR (Status);

    ");
  }
}

include_statement(AutoGen.c, "
  }

");
```

7.3.4.4 Library Destructor Statements

Contained if there are DESTRUCTORs defined in [Defines] section in INF file of the library instances that are being linked to.

```
If (DESTRUCTOR defined in INF) {

  If (MODULE_TYPE == "BASE") {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <DESTRUCTOR> (
        VOID
      );

    ");
  }

  If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <DESTRUCTOR> (
        IN EFI_PEI_FILE_HANDLE       FileHandle,
        IN EFI_PEI_SERVICES          **PeiServices
      );

    ");
  }

  If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
      (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
      (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
      (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
      (MOODULE_TYPE =='UEFI_DRIVER') ||
      (MODULE_TYPE == 'UEFI_APPLICATION')) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      <DESTRUCTOR> (
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
      );

    ");
  }

} // End DESTRUCTOR defined in INF

If (MODULE_TYPE == "BASE") {
  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryDestructorList (
      VOID
    )

  ");
}
```

```
If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryDestructorList (
      IN EFI_PEI_FILE_HANDLE       FileHandle,
      IN EFI_PEI_SERVICES          **PeiServices
    )

  ");
}

If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
    (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
    (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
    (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
    (MOODULE_TYPE =='UEFI_DRIVER') ||
    (MODULE_TYPE == 'UEFI_APPLICATION')) {
  include_statement(AutoGen.c, "
    VOID
    EFIAPI
    ProcessLibraryDestructorList (
      IN EFI_PEI_FILE_HANDLE       ImageHandle,
      IN EFI_PEI_SERVICES          **SystemTable
    )

  ");
}

include_statement(AutoGen.c, "
  {

");

If (DESTRUCTOR defined in INF) {
  If (MODULE_TYPE == "BASE") {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <DESTRUCTOR>();
      ASSERT_EFI_ERROR (Status);

    ");
  }

  If ((MODULE_TYPE == "PEI_CORE") || (MODULE_TYPE == "PEIM")) {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <DESTRUCTOR>(FileHandle, PeiServices);
      ASSERT_EFI_ERROR (Status);

    ");
  }
```

```
  If ((MODULE_TYPE == 'DXE_CORE') || (MODULE_TYPE =='DXE_DRIVER') ||
      (MODULE_TYPE == 'DXE_SMM_DRIVER') ||
      (MODULE_TYPE == 'DXE_RUNTIME_DRIVER' ||
      (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
      (MOODULE_TYPE =='UEFI_DRIVER') ||
      (MODULE_TYPE == 'UEFI_APPLICATION')) {
    include_statement(AutoGen.c, "
      EFI_STATUS  Status;

      Status = <DESTRUCTOR>(ImageHandle, SystemTable);
      ASSERT_EFI_ERROR (Status);

    ");
  }
}

include_statement(AutoGen.c, "
  }

");
```

7.3.4.5 Module Entry Point Statements

Contained if there are ENTRY_POINTs defined [Defines] section in INF file.

```
If (ENTRY_POINT defined in INF) {

  If (MODULE_TYPE == 'PEI_CORE') {
    include_statement(AutoGen.c, "
      EFI_STATUS
      <ENTRY_POINT> (
        IN CONST  EFI_SEC_PEI_HAND_OFF    *SecCoreData,
        IN CONST  EFI_PEI_PPI_DESCRIPTOR  *PpiList,
        IN VOID                           *OldCoreData
      );

      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
        IN CONST  EFI_SEC_PEI_HAND_OFF    *SecCoreData,
        IN CONST  EFI_PEI_PPI_DESCRIPTOR  *PpiList,
        IN VOID                           *OldCoreData
      )

      {
        return <ENTRY_POINT> (SecCoreData, PpiList, OldCoreData);
      }

    ");
  }
```

```
If (MODULE_TYPE == 'DXE_CORE') {
  include_statement(AutoGen.c, "
    const UINT32 _gUefiDriverRevision = 0;

    VOID
    <ENTRY_POINT> (
      IN VOID  *HobStart
      );

    VOID
    EFIAPI
    ProcessModuleEntryPointList (
      IN VOID  *HobStart
      )

    {
      <ENTRY_POINT> (HobStart);
    }

  ");
}

If (MODULE_TYPE == 'PEIM') {
  include_statement(AutoGen.c, "
    GLOBAL_REMOVE_IF_UNREFERENCED const UINT32 _gPeimRevision = 0;

  ");
  If (Number of ENTRY_POINT == 0) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        )

      {
        return EFI_SUCCESS;
      }

    ");
  }

  If (Number of ENTRY_POINT == 1) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      <ENTRY_POINT> (
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        );

      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
```

```
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        )

      {
        return <ENTRY_POINT> (FileHandle, PeiServices);
      }

    ");
  }

  If (Number of ENTRY_POINT > 1) {
    include_statement(AutoGen.c, "
      <ENTRY_POINT1> (
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        );

      <ENTRY_POINT2> (
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        );

      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
        IN EFI_PEI_FILE_HANDLE  FileHandle,
        IN EFI_PEI_SERVICES     **PeiServices
        )

      {
        EFI_STATUS  Status;
        EFI_STATUS  CombinedStatus;

        CombinedStatus = EFI_LOAD_ERROR;

        Status = <ENTRY_POINT1> (FileHandle, PeiServices);
        if (!EFI_ERROR (Status) || EFI_ERROR (CombinedStatus)) {
          CombinedStatus = Status;
        }

        Status = <ENTRY_POINT2> (FileHandle, PeiServices);
        if (!EFI_ERROR (Status) || EFI_ERROR (CombinedStatus)) {
          CombinedStatus = Status;
        }

        return CombinedStatus;
      }

    ");
  }

If (MODULE_TYPE == 'DXE_SMM_DRIVER') {
```

```
If (Number of ENTRY_POINT == 0) {
  include_statement(AutoGen.c, "
    EFI_STATUS
    EFIAPI
    ProcessModuleEntryPointList (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      )

    {
      return EFI_SUCCESS;
    }

  ");
}

If (Number of ENTRY_POINT == 1) {
  include_statement(AutoGen.c, "
    EFI_STATUS
    <ENTRY_POINT> (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      );

    static BASE_LIBRARY_JUMP_BUFFER  mJumpContext;
    static EFI_STATUS  mDriverEntryPointStatus = EFI_LOAD_ERROR;

    VOID
    EFIAPI
    ExitDriver (
      IN EFI_STATUS  Status
      )

    {
      if (!EFI_ERROR (Status) ||
           EFI_ERROR (mDriverEntryPointStatus)) {
        mDriverEntryPointStatus = Status;
      }
      LongJump (&mJumpContext, (UINTN)-1);
      ASSERT (FALSE);
    }

    EFI_STATUS
    EFIAPI
    ProcessModuleEntryPointList (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      )

    {
      if (SetJump (&mJumpContext) == 0) {
        ExitDriver (<ENTRY_POINT> (ImageHandle, SystemTable));
        ASSERT (FALSE);
```

```
        }

        return mDriverEntryPointStatus;
      }

    ");
  }
}

If ((MODULE_TYPE == 'DXE_RUNTIME_DRIVER') ||
    (MODULE_TYPE == 'DXE_DRIVER') ||
    (MODULE_TYPE == 'DXE_SAL_DRIVER') ||
    (MODULE_TYPE == 'UEFI_DRIVER') ||
    (MODULE_TYPE == 'UEFI_APPLICATION')) {
  include_statement(AutoGen.c, "
    const UINT32 _gUefiDriverRevision = 0;

  ");

  If (Number of ENTRY_POINT == 0) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
        )
      {
        return EFI_SUCCESS;
      }

    ");
  }

  If (Number of ENTRY_POINT == 1) {
    include_statement(AutoGen.c, "
      EFI_STATUS
      ${Function} (
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
        );

      EFI_STATUS
      EFIAPI
      ProcessModuleEntryPointList (
        IN EFI_HANDLE        ImageHandle,
        IN EFI_SYSTEM_TABLE  *SystemTable
        )

      {
        return <ENTRY_POINT> (ImageHandle, SystemTable);
      }

      VOID
```

```
    EFIAPI
    ExitDriver (
      IN EFI_STATUS  Status
      )
    {
      if (EFI_ERROR (Status)) {
        ProcessLibraryDestructorList (gImageHandle, gST);
      }
      gBS->Exit (gImageHandle, Status, 0, NULL);
    }

  ");
}
If (Number of ENTRY_POINT > 1) {
  include_statement(AutoGen.c, "
    <ENTRY_POINT1> (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      );

    <ENTRY_POINT2> (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      );

    EFI_STATUS
    EFIAPI
    ProcessModuleEntryPointList (
      IN EFI_HANDLE        ImageHandle,
      IN EFI_SYSTEM_TABLE  *SystemTable
      )

    {

      if (SetJump (&mJumpContext) == 0) {
        ExitDriver (<ENTRY_POINT1> (ImageHandle, SystemTable));
        ASSERT (FALSE);
      }

      if (SetJump (&mJumpContext) == 0) {
        ExitDriver (<ENTRY_POINT2> (ImageHandle, SystemTable));
        ASSERT (FALSE);
      }

      return mDriverEntryPointStatus;
    }

    static BASE_LIBRARY_JUMP_BUFFER  mJumpContext;
    static EFI_STATUS  mDriverEntryPointStatus = EFI_LOAD_ERROR;

    VOID
    EFIAPI
    ExitDriver (
      IN EFI_STATUS  Status
```

```
          )
        {
          if (!EFI_ERROR (Status) ||
               EFI_ERROR (mDriverEntryPointStatus)) {
            mDriverEntryPointStatus = Status;
          }
          LongJump (&mJumpContext, (UINTN)-1);
          ASSERT (FALSE);
        }

      ");
    }
  }

}
```

7.3.4.6 Module Unload Image Statements

The following algorithm is used to process potential UNLOAD_IMAGE statements that might be defined in the [Defines] section in the INF file.

```
If (Number of UNLOAD_IMAGE in INF == 0) {
  include_statement(AutoGen.c, "
    GLOBAL_REMOVE_IF_UNREFERENCED const UINT8 \
      _gDriverUnloadImageCount = 0;

    EFI_STATUS
    EFIAPI
    ProcessModuleUnloadList (
      IN EFI_HANDLE        ImageHandle
      )
    {
      return EFI_SUCCESS;
    }
  ");
}

If (Number of UNLOAD_IMAGE in INF == 1) {
  include_statement(AutoGen.c, "
    GLOBAL_REMOVE_IF_UNREFERENCED const UINT8 \
      _gDriverUnloadImageCount = 1;

    EFI_STATUS
    <UNLOAD_IMAGE> (
      IN EFI_HANDLE        ImageHandle
      );

    EFI_STATUS
    EFIAPI
    ProcessModuleUnloadList (
      IN EFI_HANDLE        ImageHandle
      )
    {
      return <UNLOAD_IMAGE> (ImageHandle);
    }
  ");
}
```

```
If (Number of UNLOAD_IMAGE in INF > 1) {
  include_statement(AutoGen.c, "
    GLOBAL_REMOVE_IF_UNREFERENCED const UINT8 \
      _gDriverUnloadImageCount = <NumberOfUnloadImage>;

    EFI_STATUS
    <UNLOAD_IMAGE1> (
      IN EFI_HANDLE        ImageHandle
      );

    EFI_STATUS
    <UNLOAD_IMAGE2> (
      IN EFI_HANDLE        ImageHandle
      );


    EFI_STATUS
    EFIAPI
    ProcessModuleUnloadList (
      IN EFI_HANDLE        ImageHandle
      )
    {
      EFI_STATUS  Status;

      Status = EFI_SUCCESS;

      if (EFI_ERROR (Status)) {
        <UNLOAD_IMAGE1> (ImageHandle);
      } else {
        Status = <UNLOAD_IMAGE1> (ImageHandle);
      }

      if (EFI_ERROR (Status)) {
        <UNLOAD_IMAGE2> (ImageHandle);
      } else {
        Status = <UNLOAD_IMAGE2> (ImageHandle);
      }

      return Status;
    }
  ");
}
```

7.3.4.7 Global variables

These are generated from "Guids", "Protocols", "Ppis", "xxxPcd" sections of the .inf file and .uni files.

```
InfList = [];

add (ModuleInf, InfList);

foreach LibraryInstance {
  add (LibraryInf, InfList);
  foreach DependentLibraryInstance {
    add (LibraryInf, InfList);
  }
}

foreach INF in InfList {
  If ("[Guids]" defined in INF) {
    foreach GuidCName {
      include_statement(AutoGen.c, "
        GLOBAL_REMOVE_IF_UNREFERENCED EFI_GUID <GuidCName> = <GuidValue>;
      ");
    }
  }

  If ("[Protocols]" defined in INF) {
    foreach ProtocolGuidCName {
      include_statement(AutoGen.c, "
        GLOBAL_REMOVE_IF_UNREFERENCED EFI_GUID <ProtocolGuidCName> =
          <GuidValue>;
      ");
    }
  }

  If (("[Ppis]" defined in INF) {
    foreach PpiGuidCName {
      include_statement(AutoGen.c, "
        GLOBAL_REMOVE_IF_UNREFERENCED EFI_GUID <PpiGuidCName> =
          <GuidValue>;
      ");
    }
  }

  If ("[Pcd]" defined in INF) {
    foreach PcdCName {
      If ((PcdDatumType == 'VOID*') && (PcdValue is array)) {
        include_statement(AutoGen.c, "
          GLOBAL_REMOVE_IF_UNREFERENCED UINT8 <PcdCName> =
            <PcdValueMacro>;
        ");
      } Else {
        include_statement(AutoGen.c, "
          GLOBAL_REMOVE_IF_UNREFERENCED <PcdDatumType> <PcdCName> =
            <PcdValueMacro>;
        ");
      }
    }
  }

  If (.UNI file found in INF SourcesSection) {
```

```
      include_statement(AutoGen.c, "

        unsigned char MiscSubclassStrings[] = {
          ……
        }

      ");
    }

  }
```

7.4 Auto-generated Makefiles

The actual build actions are done via "MAKE" system. This system is "nmake" in Windows environment and "make" in GCC (Linux and Mac OS/X) environment. The makefiles are composed in two levels, platform level (top-level) makefile and module level makefile. For one platform, one makefile is generated for each tool chain and build target (DEBUG/RELEASE). Each module has a makefile generated and referenced in the platform level makefile.

The build tool calls the build script tool (nmake or gmake or make) for a platform or module Makefile. In platform build mode, the build tool calls the build script tool giving the Platform Makefile as an argument. The Platform Makefile in turn launches the build script tool for each Module's Makefile.

In Module mode, the build tool calls the build script tool, giving the Module Makefile as an argument. However, in Module Mode, if the build tool target is "fds", then the build tool calls the build script tool, giving the Module Makefile as an argument. It then calls the build script tool, giving the Platform Makefile as an argument with a target of "fds".

7.4.1 Module Makefile

This section describe the formats of the individual component/module Makefiles. Users may generate a custom makefile for their EDK component or EDK II module based on the information provided by this section.

The module makefile is composed by two parts: macro definitions and target definitions.

In the pseudo-code provided, the MACRO, $(MODULE_BUILD_DIR) is constructed using the following rules:

- If the .dsc file's OUTPUT_DIRECTORY value (path) starts with an alpha character, the value of the OUTPUT_DIRECTORY statement is relative to the WORKSPACE, otherwise it is considered an absolute directory path.
- Separate Top Level Makefiles are required for each TARGET and ToolChainTag.
- The Top Level Makefile processes all makefiles in different architectures.

```
If (isalphs(getValue("OUTPUT_DIRECTORY", DscFile)[0]) {
  MOD_BUILD_DIR = "$WORKSPACE)\" +
    getValue("OUTPUT_DIRECTORY", DscFile)
} else {
  MOD_BUILD_DIR1 = getValue("OUTPUT_DIRECTORY", DscFile)
}
Foreach Target in ActiveTargetList {
  Foreach ToolChainTag in ActiveToolChain {
    MOD_BUILD_DIR2 = $(MOD_BUILD_DIR1) + "\" +
                     Target + "_" + ToolChainTag + "\";
    foreach Arch in ActiveArchList {
      MODULE_BUILD_DIR = $(MOD_BUILD_DIR2) + Arch + "\";
      MODULE_BUILD_DIR += getDirPart(InfFile) + "\";
      MODULE_BUILD_DIR += getValue("BASE_NAME", InfFIle) + "\";
      MAKEFILE = $(MODULE_BUILD_DIR) + "Makefile";
      genModuleMakefile($(MAKEFILE));
      addModuleToList($MAKEFILE, MakefileList);
    }
    genTopMakefile($(MOD_BUILD_DIR2) + "Makefile")
  }
}
```

7.4.1.1 Macro definitions

7.4.1.1.1 Platform information

These come from "Defines" section in the DSC file.

```
MakefileList = $(PLATFORM_MAKEFILE)
Foreach InfFile {
  MakefileList += $(MODULE_MAKEFILE)
}
foreach Makefile in MakefileList {
  include_statement($(MODULE_BUILD_DIR)\Makefile, "
    PLATFORM_NAME = getValue("PLATFORM_NAME", DscFile);
    PLATFORM_GUID = getValue("PLATFORM_GUID", DscFile);
    PLATFORM_VERSION = getValue("PLATFORM_VERSION", DscFile);
    PLATFORM_RELATIVE_DIR = getDirPart(ActivePlatform);
    PLATFORM_DIR = "$(WORKSPACE)\" + getDirPart(ActivePlatform);
    PLATFORM_OUTPUT_DIR = getValue("OUTPUT_DIRECTORY", DscFile);

  ");
}
```

Example:

```
PLATFORM_NAME = NT32
PLATFORM_GUID = EB216561-961F-47EE-9EF9-CA426EF547C2
PLATFORM_VERSION = 0.3
PLATFORM_RELATIVE_DIR = Nt32Pkg
PLATFORM_DIR = $(WORKSPACE)\Nt32Pkg
PLATFORM_OUTPUT_DIR = Build\NT32
```

7.4.1.1.2 Module information

These come from "Defines" section in the INF file and "Components" section in DSC file.

```
Foreach InfFile {
  include_statement($(MODULE_BUILD_DIR)\Makefile, "
    MODULE_NAME = getValue("BASE_NAME", InfFile)
    MODULE_GUID = getValue("FILE_GUID", InfFIle)
    MODULE_VERSION = getValue("VERSION_STRING", InfFile)
    MODULE_TYPE = getValue("MODULE_TYPE", InfFile);
    MODULE_FILE_BASE_NAME = getValue("BASE_NAME", InfFile)
    BASE_NAME = $(MODULE_NAME)
    MODULE_RELATIVE_DIR = getDirPart(InfFile)
    MODULE_DIR = "$(WORKSPACE)\" + getDirPart(InfFile)

  ");
}
```

Example:

```
MODULE_NAME = HelloWorld
MODULE_GUID = 6987936E-ED34-44db-AE97-1FA5E4ED2116
MODULE_VERSION = 1.0
MODULE_TYPE = UEFI_APPLICATION
MODULE_FILE_BASE_NAME = HelloWorld
BASE_NAME = $(MODULE_NAME)
MODULE_RELATIVE_DIR = MdeModulePkg\Application\HelloWorld
MODULE_DIR = $(WORKSPACE)\MdeModulePkg\Application\HelloWorld
```

7.4.1.1.3 Build configuration

These come from $(WORKSPACE)/Conf/target.txt, command line options, or "Defines" section in DSC file.

```
ARCH = IA32
TOOLCHAIN_TAG = MYTOOLS
TARGET = DEBUG
```

7.4.1.1.4 Build directories

These are determined by build tools. Macro "DEST_DIR_OUTPUT" and "DEST_DIR_DEBUG" are generated for backward compatibility.

```
PLATFORM_BUILD_DIR = $(WORKSPACE)\Build\NT32
BUILD_DIR = $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS
BIN_DIR = $(BUILD_DIR)\IA32
LIB_DIR = $(BIN_DIR)
MODULE_BUILD_DIR = \
  $(BUILD_DIR)\IA32\MdeModulePkg\Application\HelloWorld\HelloWorld
OUTPUT_DIR = $(MODULE_BUILD_DIR)\OUTPUT
DEBUG_DIR = $(MODULE_BUILD_DIR)\DEBUG
DEST_DIR_OUTPUT = $(OUTPUT_DIR)
DEST_DIR_DEBUG = $(DEBUG_DIR)
```

7.4.1.1.5 Tools flags,

These are used to concatenate the flags from different places in the predefined order. The order makes sure that the flags defined DSC file can override flags in INF file and default ones. Note that in the code example below, the tools will expand the values into a single line - $(TOOLS_DEF_LZMA_FLAGS) does not appear in the Makefile, only the flag values appear.

```
LZMA_FLAGS = $(TOOLS_DEF_LZMA_FLAGS) $(INF_LZMA_FLAGS) \
  $(DSC_LZMA_FLAGS) $(DSC_INF_LZMA_FLAGS)
PP_FLAGS = $(TOOLS_DEF_PP_FLAGS) $(INF_PP_FLAGS) \
  $(DSC_PP_FLAGS) $(DSC_INF_PP_FLAGS)
SLINK_FLAGS = $(TOOLS_DEF_SLINK_FLAGS) $(INF_SLINK_FLAGS) \
  $(DSC_SLINK_FLAGS) $(DSC_INF_SLINK_FLAGS)
CC_FLAGS = $(TOOLS_DEF_CC_FLAGS) $(INF_CC_FLAGS) \
  $(DSC_CC_FLAGS) $(DSC_INF_CC_FLAGS)
APP_FLAGS = $(TOOLS_DEF_APP_FLAGS) $(INF_APP_FLAGS) \
  $(DSC_APP_FLAGS) $(DSC_INF_APP_FLAGS)
VFRPP_FLAGS = $(TOOLS_DEF_VFRPP_FLAGS) $(INF_VFRPP_FLAGS) \
  $(DSC_VFRPP_FLAGS) $(DSC_INF_VFRPP_FLAGS)
DLINK_FLAGS = $(TOOLS_DEF_DLINK_FLAGS) $(INF_DLINK_FLAGS) \
  $(DSC_DLINK_FLAGS) $(DSC_INF_DLINK_FLAGS)
ASM_FLAGS = $(TOOLS_DEF_ASM_FLAGS) $(INF_ASM_FLAGS) \
  $(DSC_ASM_FLAGS) $(DSC_INF_ASM_FLAGS)
TIANO_FLAGS = $(TOOLS_DEF_TIANO_FLAGS) $(INF_TIANO_FLAGS) \
  $(DSC_TIANO_FLAGS) $(DSC_INF_TIANO_FLAGS)
MAKE_FLAGS = $(TOOLS_DEF_MAKE_FLAGS) $(INF_MAKE_FLAGS) \
  $(DSC_MAKE_FLAGS) $(DSC_INF_MAKE_FLAGS)
ASMLINK_FLAGS = $(TOOLS_DEF_ASMLINK_FLAGS) $(INF_ASMLINK_FLAGS)\
  $(DSC_ASMLINK_FLAGS) $(DSC_INF_ASMLINK_FLAGS)
ASL_FLAGS = $(TOOLS_DEF_ASL_FLAGS) $(INF_ASL_FLAGS) \
  $(DSC_ASL_FLAGS) $(DSC_INF_ASL_FLAGS)
```

7.4.1.1.6 Tools path,

These come from the file specified by "TOOL_CHAIN_CONF" definition in $(WORKSPACE)/Conf/target.txt.

```
LZMA = H:\dev\AllPackagesDev\IntelRestrictedTools\Bin\Win32\LzmaCompress.exe
PP = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\cl.exe
SLINK = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\lib.exe
CC = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\cl.exe
APP = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\cl.exe
VFRPP = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\cl.exe
DLINK = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\link.exe
ASM = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\ml.exe
TIANO = TianoCompress.exe
MAKE = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\nmake.exe
ASMLINK = C:\WINDDK\3790.1830\bin\bin16\link.exe
ASL = C:\ASL\iasl.exe
```

7.4.1.1.7 Shell commands,

These are used to make sure that the file operations for both nmake and GNU make system become as the same as possible.

```
# shell commands for nmake
RD = rmdir /s /q
RM = del /f /q
MD = mkdir
CP = copy /y
MV = move /y

# shell commands for gnu make
RD = rm -r -f
RM = rm -f
MD = mkdir -p
CP = cp -u -f
MV = mv -f
```

7.4.1.1.8 Source files and target files list macro

In these, "*<FILE_TYPES>*" macros are generated from $(WORKSPACE)/Conf/build_rule.txt and files listed in "Sources" section in INF file, "*INC*" macro is generated from "Includes" section in DEC file and "Packages" section in INF file, "*LIBS*" macro is generated from "LibraryClasses" section in INF file and DSC file, and "*COMMON_DEPS*" macro is generated by parsing recursively the "*#include*" preprocessor directives in source code files.

```
C_CODE_FILES = \
  $(WORKSPACE)\MdeModulePkg\App\Hello\HelloWorld.c
DYNAMIC_LIBRARY_FILE_LIST = $(DEBUG_DIR)\$(MODULE_NAME).dll
UNKNOWN_TYPE_FILE_LIST = $(DEBUG_DIR)\$(MODULE_NAME).efi
OBJECT_FILE_LIST = $(OUTPUT_DIR)\.\HelloWorld.obj
STATIC_LIBRARY_FILE_LIST = $(OUTPUT_DIR)\$(MODULE_NAME).lib

INC = <include search path list>
LIBS = <dependent library file list>
COMMON_DEPS = <header file list>
```

7.4.1.1.9 Target macros

In these "**CODA_TARGET**" is generated according to the last rule(s) in rule chains defined in $(WORKSPACE)/Conf/build_rule.txt.

```
INIT_TARGET = init
CODA_TARGET = $(DEBUG_DIR)\$(MODULE_NAME).efi
```

7.4.1.2 Target definitions

7.4.1.2.1 “all” target

Default target which actually executes against the "mbuild" target.

7.4.1.2.2 “pbuild” target

Target which is used to build the source files of current module only. It's always used in top-level makefile because the libraries will be built above all non-library modules.

```
pbuild: $(INIT_TARGET) $(CODA_TARGET)
```

7.4.1.2.3 “mbuild” target

Actual default target which is used for single module build mode. Because in single

module build mode the top-level make file will not be called, the build system has to build libraries that the current module needs in module's makefile. "`mbuild`" target is used for this purpose.

```
mbuild: $(INIT_TARGET) gen_libs $(CODA_TARGET)
gen_libs:
     cd $(BUILD_DIR)\IPF\MdePkg\Library\DxePcdLib\DxePcdLib && "$(MAKE)"
$(MAKE_FLAGS)
     cd $(BUILD_DIR)\IPF\MdePkg\Library\BaseLib\BaseLib && "$(MAKE)"
$(MAKE_FLAGS)
     cd $(BUILD_DIR)\IPF\CsiCpuUncorePkg\Library\ItcTimerLib\ItcTimerLib &&
"$(MAKE)" $(MAKE_FLAGS)
     cd $(MODULE_BUILD_DIR)
```

7.4.1.2.4 “fds” target

Target used to re-generate a flash image after a single module build. It calls top-level makefile to do this.

```
fds: mbuild gen_fds
gen_fds:
     cd $(BUILD_DIR) && "$(MAKE)" $(MAKE_FLAGS) fds
     cd $(MODULE_BUILD_DIR)
```

7.4.1.2.5 “init” target

Target used to print verbose information and create necessary directories used for build.

```
init:
     -@echo Building ... $(MODULE_NAME) $(MODULE_VERSION) [$(ARCH)] in platform
$(PLATFORM_NAME) $(PLATFORM_VERSION)
     -@if not exist $(DEBUG_DIR) mkdir $(DEBUG_DIR)
     -@if not exist $(OUTPUT_DIR) mkdir $(OUTPUT_DIR)
```

7.4.1.2.6 Miscellaneous build targets

Targets which are used to build source files to object files and then in turn into final lib file, efi file or other files. These targets are generated according to the rule chains in $(WORKSPACE)/Conf/build_rule.txt. For example:

```
$(OUTPUT_DIR)\.\ModuleFile.obj : $(COMMON_DEPS)
     "$(CC)" /Fo$(OUTPUT_DIR)\.\ModuleFile.obj $(CC_FLAGS) $(INC) \
     $(WORKSPACE)\MyPlatformPkg\MySubDir\ModuleFile.c

$(OUTPUT_DIR)\$(MODULE_NAME).lib : $(OBJECT_FILE_LIST)
     "$(SLINK)" $(SLINK_FLAGS) \
     /OUT:$(OUTPUT_DIR)\$(MODULE_NAME).lib $(OBJECT_FILE_LIST)

$(DEBUG_DIR)\$(MODULE_NAME).dll : \
     $(OUTPUT_DIR)\$(MODULE_NAME).lib $(LIBS) $(MAKE_FILE)
     "$(DLINK)" /OUT:$(DEBUG_DIR)\$(MODULE_NAME).dll $(DLINK_FLAGS)\
     $(DLINK_SPATH) $(LIBS) $(OUTPUT_DIR)\$(MODULE_NAME).lib

$(DEBUG_DIR)\$(MODULE_NAME).efi : $(DEBUG_DIR)\$(MODULE_NAME).dll
     GenFw -e $(MODULE_TYPE) -o $(DEBUG_DIR)\$(MODULE_NAME).efi \
     $(DEBUG_DIR)\$(MODULE_NAME).dll
     $(CP) $(DEBUG_DIR)\$(MODULE_NAME).efi $(OUTPUT_DIR)
     $(CP) $(DEBUG_DIR)\$(MODULE_NAME).efi $(BIN_DIR)
     -$(CP) $(DEBUG_DIR)\*.map $(OUTPUT_DIR)

$(OUTPUT_DIR)\.\AutoGen.obj : \
$(WORKSPACE)\Build\MyPlatform\DEBUG_ICC\IPF\MyPlatformPkg\MyModDir\MyModDir\DEBU
G\AutoGen.c
    "$(CC)" /Fo$(OUTPUT_DIR)\.\AutoGen.obj $(CC_FLAGS) $(INC) \
$(WORKSPACE)\Build\MyPlatform\DEBUG_ICC\IPF\MyPlatformPkg\MyModDir\MyModDir\DEBU
G\AutoGen.c
```

7.4.1.2.7 clean, cleanall, cleanlib

Targets used to delete part or all files generated during build.

```
clean:
     if exist $(OUTPUT_DIR) rmdir /s /q $(OUTPUT_DIR)

cleanall:
     if exist $(DEBUG_DIR) rmdir /s /q $(DEBUG_DIR)
     if exist $(OUTPUT_DIR) rmdir /s /q $(OUTPUT_DIR)
     del /f /q *.pdb *.idb > NUL 2>&1

cleanlib:
     cd $(BUILD_DIR)\IPF\MdePkg\Library\DxePcdLib\DxePcdLib && \
      "$(MAKE)" $(MAKE_FLAGS) cleanall
     cd $(BUILD_DIR)\IPF\MdePkg\Library\BaseLib\BaseLib && \
      "$(MAKE)" $(MAKE_FLAGS) cleanall
     cd $(BUILD_DIR)\IPF\CsiCpuUncorePkg\Library\ItcTimerLib\ItcTimerLib &&
"$(MAKE)" $(MAKE_FLAGS) cleanall
     cd $(MODULE_BUILD_DIR)
```

7.4.2 Top level Makefile

The top-level makefile is composed by following macro and target definitions.

7.4.2.1 Macro Definitions

Platform information macros come from "Defines" section in the DSC file.

```
PLATFORM_NAME = NT32
PLATFORM_GUID = EB216561-961F-47EE-9EF9-CA426EF547C2
PLATFORM_VERSION = 0.3
PLATFORM_DIR = $(WORKSPACE)\Nt32Pkg
PLATFORM_OUTPUT_DIR = Build\NT32
```

Build configuration macros come from $(WORKSPACE)/Conf/target.txt, command line options, and/or "Defines" section in the DSC file.

```
TOOLCHAIN_TAG = MYTOOLS
TARGET = DEBUG
```

Build directories are determined by the build tools.

```
BUILD_DIR = $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS
FV_DIR = $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\FV
```

7.4.2.2 Build Target Definitions

7.4.2.2.1 "all" target

The default target. It determines the build order of parts consisting of a platform.

```
all: init build_libraries build_modules build_fds
```

7.4.2.2.2 "init" target

Performs initialization work such as creating directories.

```
init:
  -@echo Building ... $(PLATFORM_NAME) $(PLATFORM_VERSION) [IA32]
  -@if not exist $(BUILD_DIR)\IA32 mkdir $(BUILD_DIR)\IA32
  -@if not exist $(FV_DIR) mkdir $(FV_DIR)
```

7.4.2.2.3 "libraries" target

This is used to build all libraries only.

```
libraries: init build_libraries

build_libraries:
  cd $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseLib\BaseLib
&& "$(MAKE)" $(MAKE_FLAGS) pbuild
  cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseMemoryLibOptDxe\Ba
seMemoryLibOptDxe && "$(MAKE)" $(MAKE_FLAGS) pbuild
  cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BasePcdLibNull\BasePcd
LibNull && "$(MAKE)" $(MAKE_FLAGS) pbuild
      ......
```

7.4.2.2.4 "modules" target

This is used to build all modules and the libraries they depend on.

modules: init build_libraries build_modules

```
build_modules:
       cd $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\Nt32Pkg\Sec\SecMain &&
"$(MAKE)" $(MAKE_FLAGS) pbuild
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdeModulePkg\Core\Pei\PeiMain &&
"$(MAKE)" $(MAKE_FLAGS) pbuild
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdeModulePkg\Universal\PCD\Pei\Pcd &&
"$(MAKE)" $(MAKE_FLAGS) pbuild
   ......
```

7.4.2.2.5 "fds" target

This is used to generated flash image of the platform.

```
fds: init build_fds

build_fds:
       -@echo Generating flash image, if any ...
       GenFds -f H:\dev\AllPackagesDev\Nt32Pkg\Nt32Pkg.fdf -o $(BUILD_DIR) -t
$(TOOLCHAIN_TAG) -b $(TARGET) -p H:\dev\AllPackagesDev\Nt32Pkg\Nt32Pkg.dsc -a
IA32
```

7.4.2.2.6 "clean", "cleanall" and "cleanlib" targets

These are used to clean part of all the files generated during platform build.

```
clean:
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseTimerLibNullTempla
te\BaseTimerLibNullTemplate && "$(MAKE)" $(MAKE_FLAGS) clean
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseDebugLibNull\BaseD
ebugLibNull && "$(MAKE)" $(MAKE_FLAGS) clean
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseLib\BaseLib &&
"$(MAKE)" $(MAKE_FLAGS) clean
       ......

cleanlib:
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseTimerLibNullTempla
te\BaseTimerLibNullTemplate && "$(MAKE)" $(MAKE_FLAGS) cleanall
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseDebugLibNull\BaseD
ebugLibNull && "$(MAKE)" $(MAKE_FLAGS) cleanall
       cd
$(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS\IA32\MdePkg\Library\BaseLib\BaseLib &&
"$(MAKE)" $(MAKE_FLAGS) cleanall
       ......

cleanall:
       if exist $(BUILD_DIR)\IA32 rmdir /s /q $(BUILD_DIR)\IA32
       if exist $(FV_DIR) rmdir /s /q $(FV_DIR)
```

7.4.2.2.7 "run" target

This special target is used to execute the emulation platform such as NT32. It should never be used for real platforms.

```
run:
        cd $(BUILD_DIR)\IA32
        SecMain
        cd $(BUILD_DIR)
```

7.5 Binary Modules

EDK II accommodates distribution of binary module code for inclusion into a firmware volume. This feature is used by vendors who have a proprietary code base, but need to provide their customers with the ability to use that code in a platform. Vendors may protect their IP by distributing only module code in either lib, bin, or efi format, without distributing debug files or sources.

A binary module must have a "binaries" section. Its INF file should not be put in DSC file and the build tools should not try to do module build for a binary module, unless there's "LIB" type of files declared. The INF file of a binary module is always put in FDF file for flash image generation. The binary files can also be referenced directly in FDF. Please refer to Section 9 for details.

Binary modules are used only with FDF files.

8
$(MAKE) Stage

This chapter describes the processing of the source files into EFI files.

The make stage starts out by building required libraries, followed by the EDK components and finally, EDK II modules. The outputs of this stage are linked PE32+/ COFF images that have been processed to replace the standard header with an appropriate EFI header.

How a file will be handled is defined in `$(WORKSPACE)/Conf/build_rule.txt`, and converted into actions and targets in makefile.

How a file will be processes is defined in the file specified by the BUILD_RULE_CONF statement in target.txt or the default file `$(WORKSPACE)/Conf/build_rule.txt`. In the previous step, those rules were used to generate makefiles.

8.1 Overview

From a platform point of view, what will be done in $(MAKE) stage includes building library modules, building non-library modules and finally generating flash image(s).

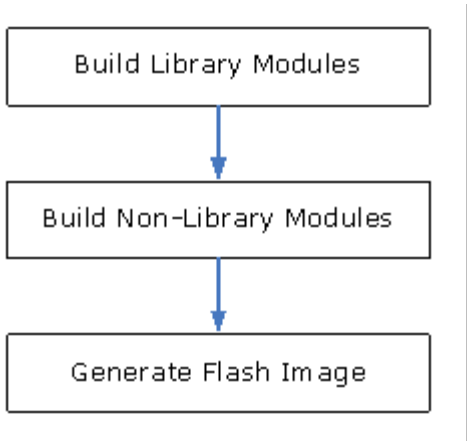


Figure 20. EDK II Build Process - Platform Point of View (PoV)

From a module point of view, things done in $(MAKE) stage includes preprocessing, compiling or assembling, static/dynamic linking and module image generation.

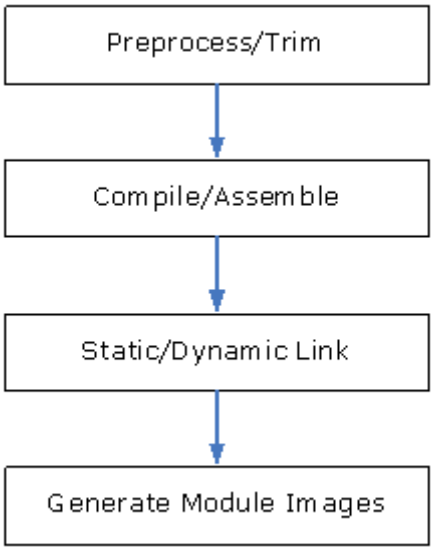


Figure 21. EDK II Build Process - Module PoV

8.2 Preprocess/Trim

Preprocessing is an intrinsic behavior of C compiler and will be always done automatically without explicitly calling. A separate "preprocessing" step is needed by those non-C files which have no preprocessing supported in their compiler or assembler.

For example, in order to use macros defined in C header files, "#include" directives can be used in an assembly file. A separated "preprocessing" step will be used to complete macro replacement before calling assembler. VFR files, .asl files and .dxs files also need "preprocessing" step to allow using macros in them.

In addition, the preprocessed assembly files, VFR files and .dxs files need an additional "Trim" step to remove unnecessary contents left by preprocessor.

8.3 Compile/Assembly

For C and assembly files, the usual C compiler or assembler is used to generate object files for them.

For VFR files, there's a special VfrCompiler tool used to generate C and header files from them, then the standard compiler is used to generate object files.

There's a special C file with .aslc extension. The standard C compiler is used to generate object files.

For ASL files, the ASL compiler is used to generate ACPI machine language files.

8.4 Static Link

Static link step is used for all modules with C files. For library modules, linking all object files into static library file is the last step. A static link step for non-library modules is not necessary to generate the final image file; however, for better optimization purpose for MSFT tool chains this step is included.

For those modules with no C files, the static link step is skipped.

8.5 Dynamic Link

Dynamic link step is used for non-library modules which have .c files, .aslc files and/or .asm16 (real mode assembly) files declared in their INF files.

The static library file generated in static link step will be linked (DLINK) together with other static library files generated from dependent library modules into .dll file.

Object files generated from ASLC files will be linked (DLINK) to .dll file directly without static link step.

Object files generated from real mode assembly files are linked to .com files by real mode linker (ASMLINK).

8.6 Generate Module Images

The final images generated by building a module are files which can be recognized by EFI/Framework protocols. The types of those files supported by default are EFI executable image file (.efi), ACPI machine language file (.aml), ACPI table file (.acpi), real mode executable file (.com) and macrocode binary file (.bin).

The .efi file will be generated for non-library modules which have C files declared. It's converted from .dll file created during the dynamic link step by the GenFw tool. Also:

- .aml file is generated from .asl file in Compile/Assembly step.
- .acpi file is converted from .dll file by the GenFw tool.
- .com file is generated in the Dynamic Link step by real mode linker (ASMLINK).
- .bin file is converted from .txt file by GenFw tool.

8.6.1 GenFw

This tool is used to generate UEFI Firmware Image files based on Component or Module types listed in the INF files from the PE/PE32+/COFF images generated by the third party tool chains. This takes .dll files created during the compile portion of the $(MAKE) stage, converting the header and creating the .efi files. Additional functions of the GenFw tool are discussed in Section 9.1 and Section 9.2.

8.7 Generate Platform Images

The final images generated by building a platform are always FVs or FDs if an FDF file is declared in platform's DSC file. The GenFds tool is used for this purpose. This is the final step of building a platform, which is called in top-level makefile. For details regarding GenFds please refer to Section 9.1, Section 9.2 and Section 9.3.

9 ImageGen Stage - FLASH Images

This chapter describes the processing of the EFI files generated by the $(MAKE) Stage into FLASH binary images. Some of the PCDs defined or used in conditional directives in the FDF are set in the platform's DSC file. The tools must make at least one pass over the DSC file to get PCD values for conditional directives and other PCD entries used in the FDF file. If a Feature Flag PCD value, used in a conditional directive, cannot be determined the build should break.

9.1 Overview of Flash Device Layout

The GenFds tool is typically called as the last step of a platform build's $(MAKE) Stage. The build.exe command will setup the call to GenFds and let the Make utility call the program. The GenFds program can also be executed by the developer from the command line.

In order to execute from the command-line, the tool needs to have it's environment setup. The following is an example of executing GenFds as a stand-alone command.

```
GenFds -f $(WORKSPACE)\Nt32Pkg\Nt32Pkg.fdf \
  -o $(WORKSPACE)\Build\NT32\DEBUG_MYTOOLS -t MYTOOLS -b DEBUG -v \
  -p $(WORKSPACE)\Nt32Pkg\Nt32Pkg.dsc -a IA32
```

GenFds calls several other tools during the generation of an FD image:

GenSec

This application is used to generate valid **EFI_SECTION** type files from PE32/PE32+/ COFF image files or other binary files. The utility will attach a valid section or PEIM header to the input file as defined in the PI specification.

GenFfs

This application is used to generate FFS files for inclusion in a firmware volume. Rules specified in the FDF file stipulate how the FFS file will be organized (what kind of sections should reside in it and in what format).

GenFv

This application is used to generate FV image by taking what and how to place FFS into it from the corresponding FV.inf file.

GenFw

This application is used to generate UEFI Firmware Image files based on Component or Module types listed in the INF files from the PE/PE32+/COFF images generated by the third party tool chains.

GenVtf

This application generates the Boot Strap File (AKA Volume Top File, or VTF) for IA32 X64, and IPF images.

Figure 22 shows how the above tools involved in the GenFds process.

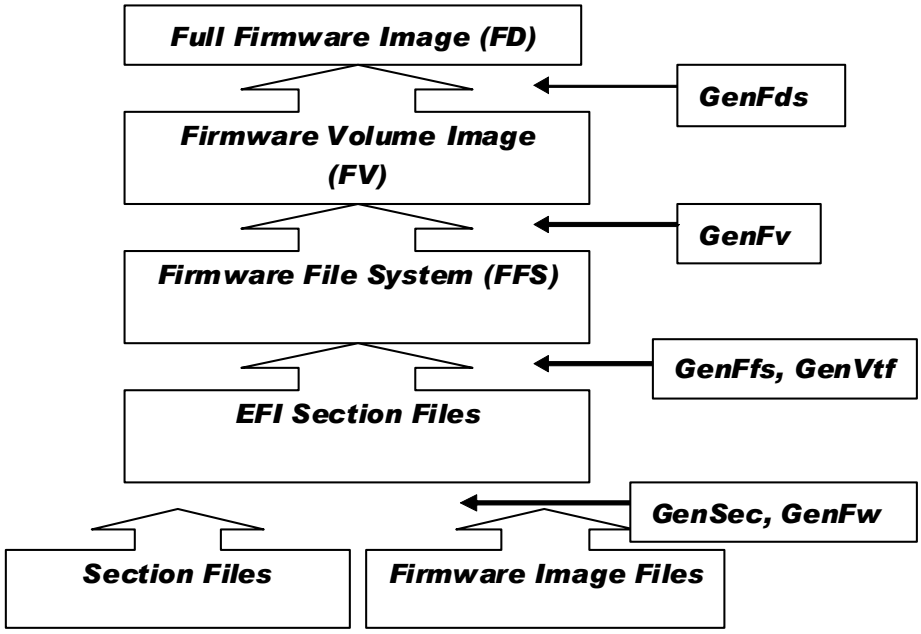


Figure 22. FD image generation process

9.2 Parsing FDF Meta-Data File

GenFds get the flash image organization information from the FDF file which is specified in command line with the '-f' option. Files that comprise the flash image are described by INF and FILE statements in FV sections of an FDF file. These files contain file name, file type and other useful information that let GenFds know which rule specified in FDF file should be used to generate the FFS file. The location of the output directory containing the image files created by GenFw or provided as binary images from $(MAKE) stage is described in the .DSC file which is specified in command line using the '-p' option. As more than one architecture might be supported by the platform, the '-a' option clarifies outputs for the different architectures.

INF files are listed in a FV in the FDF file following the format in the example:

```
INF  LakeportX64Pkg/SecCore/SecCore.inf
INF  MdeModulePkg/Core/Pei/PeiMain.inf
INF  MdeModulePkg/Universal/PCD/Pei/Pcd.inf
INF  IntelFrameworkModulePkg/Universal/StatusCode/Pei/PeiStatusCode.inf
INF  IntelFrameworkModulePkg/Universal/VariablePei/VariablePei.inf
```

All the paths in the above example are relative to the $(Workspace) directory. The information in the INF files determine how an FFS will be generated. If you want to customize the FFS, you can specify an override by adding an override in the INF:

```
INF  RuleOverride = PICOMPRESSED Ich7Pkg/UhciPei/Ich7Uhci.inf
INF  RuleOverride = PICOMPRESSED \
       My/Bus/Pci/UhciPei/UhciPei.inf
INF  RuleOverride = PICOMPRESSED \
       My/Bus/Usb/UsbBusPei/UsbBusPei.inf
INF  RuleOverride = PICOMPRESSED \
       My/Bus/Usb/UsbBotPei/UsbBotPei.inf
INF  RuleOverride = PICOMPRESSED \
       My/Bus/Isa/IsaFloppyPei/IsaFloppyPei.inf
INF  RuleOverride = PICOMPRESSED \
       My/Universal/Disk/FileSystem/FatPei/FatPei.inf
```

In the above example, GenFds will use the "PICOMPRESSED" rule to generate the FFS regardless of the rules that would normally process the INF file.

9.2.1 FILE Format Example

If the file you want to place into flash is not built using information from an INF file (for example, a micro-code that should be placed into FV), the file can be directly specify using FILE statement. The following is an example of the FILE format:

```
FILE DRIVER = 961578FE-B6B7-44c3-AF35-6BC705CD2B1F {
    SECTION PE32 = FatBinPkg/EnhancedFatDxe/X64/Fat.efi
  }
```

In this example, the Fat.efi file is placed into a PE32 section first and then placed into the generated 'DRIVER' FFS "named" with the specified GUID.

9.3 Build Intermediate Images

9.3.1 Binary modules

Binary modules can be inserted into flash image in one of two ways. The first way is to use the FILE statement mentioned in Section 9.2.1. The second way uses an INF file that describes binary files, like the one below:

```
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = Logo
  FILE_GUID                      = 7BB28B99-61BB-11D5-9A5D-0090273FC14D
  MODULE_TYPE                    = USER_DEFINED
  VERSION_STRING                 = 1.0
  EDK_RELEASE_VERSION            = 0x00020000
  EFI_SPECIFICATION_VERSION      = 0x00020000

[Binaries.common]
  BIN|Logo.bmp|*
```

This inf file shows that binary file Logo.bmp will be wrapped into the Logo FFS file. This kind of INF file is specified using standard INF statement in an FV section of the FDF file.

9.3.2 Creating EFI Sections

Sections are produced by GenSec tool using information in FDF file of what type and content the section should contain. Section information in FDF file belongs to two

categories: either it is a leaf section, or it is an encapsulate section. Encapsulation sections may contain one or more leaf sections or other encapsulate section. The leaf section information appears in the FILE statement in Section 9.2.1, the PE32 section type for the Fat.efi file. Normally this information is enough for GenSec tool, however, more information can be specified by specifying a RULE section in the FDF file. Rules in an FDF file, look like:

```
[Rule.Common.SEC]
  FILE SEC = $(NAMED_GUID)      {
    TE  TE    Align = 8       |.efi
    RAW BIN   Align = 16      |.com
  }
```

The above rule stipulates that for file type "SEC" in all build architectures, the generated FFS should contain one TE section with 8-byte alignment and one RAW section with 16-byte alignment.

Different information can be specified for different section types:

```
[Rule.Common.PEIM]
  FILE PEIM = $(NAMED_GUID) {
     PEI_DEPEX PEI_DEPEX Optional        |.depex
     TE        TE        |.efi
     UI        STRING="$(MODULE_NAME)" Optional
     VERSION   STRING="$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
  }
```

The above rule stipulates that for file type "PEIM" in all build architectures, the generated FFS may contain at most one optional PEI_DEPEX section, must contain one TE section, and may contain at most one UI section with the UI string set to the INF file's module name, and at most one VERSION section.

9.3.3 Create an Apriori File

Some firmware volumes may require, an **APRIORI** file to be created. An **APRIORI** file is a "text" file containing a GUID-named list of two or more modules in the firmware volume. The modules will be invoked or dispatched in the order they appear in the **APRIORI** file. Only one of each PEI and DXE Apriori file is permitted within a single Firmware Volume. Nested Firmware Volumes are permitted, so Apriori files are limited to specifying the files (and not FVs) that are within the scope of the FV image in which it is located. It is permissible for nested FV images to have one PEI and one DXE Apriori file per FV. Scoping is accomplished using the curly "{}" braces.

The following example demonstrates an example of multiple **APRIORI** files.

```
[Fv.Root]
    DEFINE NT32 = $(WORKSPACE)/EdkNt32Pkg
    DEFINE BuildDir = $(OUTPUT_DIRECTORY)/$(PLATFORM_NAME)/
$(TARGET)_$(TOOL_CHAIN_TAG)

    APRIORI DXE {
        FILE DXE_CORE = B5596C75-37A2-4b69-B40B-72ABD6DD8708 {
            SECTION COMPRESS {
                SECTION PE32 = $(BuildDir)/X/Y/Z/B5596C75-37A2-4b69-B40B-
72ABD6DD8708-DxeCore.efi
                SECTION VERSION "1.2.3"
            }
        }
        INF VERSION = "1" ${NT32)/Dxe/WinNtThunk/Cpu/Cpu.inf
    }

    FILE FV_IMAGE = EF41A0E1-40B1-481f-958E-6FB4D9B12E76 {
        SECTION GUIDED 3EA022A4-1439-4ff2-B4E4-A6F65A13A9AB {
            SECTION FV_IMAGE = Dxe {
                APRIORI DXE {
                    INF a/a/a.inf
                    INF a/c/c.inf
                    INF a/b/b.inf
                }
                INF a/d/d.inf
                …
            }
        }
    }
```

In the example above, there are three FFS files in the Fv.Root and one Encapsulated FV image. The build tools will create an `APRIORI` file that will dispatch the `DXE_CORE` first, then the CPU module second. In the FV image, named by the GUID `EF41A0E`..., there will be at least five FFS files, the `APRIORI` file, named Dxe, listing the GUID names of modules a.inf, c.inf and b.inf, which will be dispatched in this order. Once complete, the d.inf module may be dispatched.

9.3.4 Create FFS Files from Leaf Sections

Section 9.2 shows the INF and FILE statements in an FDF to describe FFS files that will be placed into FV. The FILE statement is straight forward, letting you know how an FFS file is organized, as it contains section information within its scope. INF statement, on the other hand, will use a particular RULE that is determined by the module type in the INF and specified build architecture.

The `[Rule]` section of the FDF file is used to define custom rules. Custom rules may also be applied to a given INF file listed in an `[FV]` section. The `[Rule]` section is also used to define rules for module types that permit the user to define the content of the FFS file - when an FFS type is not specified by either PI or UEFI specifications.

The Rules can have multiple modifiers as shown below.

`[Rule.$(ARCH).$(MODULE_TYPE).$(TEMPLATE_NAME)]`

If no `$(TEMPLATE_NAME)` is given then the match is based on `$(ARCH)` and `$(MODULE_TYPE)` modifiers. The `$(TEMPLATE_NAME)` must be unique to the `$(ARCH)` and `$(MODULE_TYPE)`. It is permissible to use the same `$(TEMPLATE_NAME)` for two or more `[Rule]` sections only if

the $(ARCH) and the $(MODULE_TYPE) listed are different for each of the sections.

A [Rule] section is terminated by another section header or the end of file.

The content of the [Rule] section is based on the FILE and section grammar of the FV section. The difference is the FILE referenced in the [RULE] is a MACRO. The section grammar is extended to include an optional argument, *Optional*. The *Optional* argument is used to say a section is optional, that is to say if it does not exist it's O.K.

The generic form of the entries for leaf sections is:

```
<SectionType> <FileType> [Options] [{<Filename>} {<Extension>}]
```

When processing the FDF file, the rules apply in the following order:

1. If <SectionType> not defined or not a legal name, then error
2. If <FileType> not defined or not a legal name, then error
3. If [FilePath/FileName], then:
   a Add one section to FFS with a section type of <SectionType>
4. Else:
   a Find all files defined by the INF file whose file type is <FileType> and add each one to the FFS with a section type of <SectionType>
   b Add files defined in [sources] followed by files defined in [binaries]
5. If more than 1 UI section in the final FFS file, then error
6. If more than 1 VER section in the final FFS file, then error
7. If more than 1 DXE_DEPEX section in final the FFS file, then error
8. If more than 1 PEI_DEPEX section in the final FFS file, then error
9. If more than 1 SMM_DEPEX section in the final FFS file, then error.

9.3.5 Create Encapsulation Sections

There are two types of encapsulation sections, a COMPRESSION section and the GUIDED section. A COMPRESSION section uses standard UEFI compression/decompression mechanisms. Other compression schemes must use the GUIDED form of encapsulation section.

The COMPRESS encapsulation section uses the following format.

```
SECTION COMPRESS [type] {
    SECTION EFI_SECTION_TYPE = FILENAME
    SECTION EFI_SECTION_TYPE = "string"
}
```

The [type] argument is optional, only EFI_STANDARD_COMPRESSION is supported by the PI specification. The current EDK enumerations for compression are a violation of the PI specification, and SECTION GUIDED should be used instead.

The EFI_SECTION_TYPE and *FILENAME* are required sub-elements within the compression encapsulation section. for most sections, however both the VERSION (EFI_SECTION_VERSION) and UI (EFI_SECTION_USER_INTEFACE) may specify a string that will be used to create an EFI section.

The GUIDED encapsulation section uses one of the following formats.

```
SECTION GUIDED $(GUID_CNAME) [auth] {
    SECTION EFI_SECTION_TYPE = FILENAME
    SECTION EFI_SECTION_TYPE = "string"
}

SECTION GUIDED $(GUID_CNAME) [auth] FILENAME
```

The required argument is the `GUIDED` name followed by an optional "`auth`" flag. If the argument "`auth`" flag is specified, then the attribute `EFI_GUIDED_SECTION_AUTH_STATUS_VALID` must be set.

For statements that do not use a scoping notation, (the second `SECTION` statement of the two listed above), if `FILENAME` exists, the attribute `EFI_GUIDED_SECTION_PROCESSING_REQUIRED` must be set to `TRUE`. The file pointed to by `FILENAME` is the data. If `FILENAME` does not exist `EFI_GUIDED_SECTION_PROCESSING_REQUIRED` is cleared and normal leaf sections must be used.

GenSec tool uses information from these encapsulated section definition as input parameters to generate the corresponding section format.

9.4 Create the FV Image File(s)

Once all of the EFI FFS files have been created, these images are bundled into an FV image.

GenFv needs two kinds of information about the target FV:

- The FV attributes
- The list of one or more files that will be placed into this FV.

This information is defined in the FV section of the FDF file. The following example is for a FV section named "BiosUpdate."

```
[FV.BiosUpdate]
BlockSize          = 0x10000
FvAlignment        = 16
ERASE_POLARITY     = 1
MEMORY_MAPPED      = TRUE
STICKY_WRITE       = TRUE
LOCK_CAP           = TRUE
LOCK_STATUS        = TRUE
WRITE_DISABLED_CAP = TRUE
WRITE_ENABLED_CAP  = TRUE
WRITE_STATUS       = TRUE
WRITE_LOCK_CAP     = TRUE
WRITE_LOCK_STATUS  = TRUE
READ_DISABLED_CAP  = TRUE
READ_ENABLED_CAP   = TRUE
READ_STATUS        = TRUE
READ_LOCK_CAP      = TRUE
READ_LOCK_STATUS   = TRUE

FILE FV_IMAGE = EDBEDF47-6EA3-4512-83C1-70F4769D4BDE {
       SECTION GUIDED {
          SECTION FV_IMAGE = BiosUpdateCargo
       }
     }
```

This FV is very simple; it contains only one "FILE". But this file contains an entire FV image named "BiosUpdateCargo" which must be available when GenFds creates the BiosUpdate FV.

The GenFds tool will process the FDF file and place the FV attributes and contents in to an INF file (in this example, the BiosUpdate.inf file) and then processing is transferred to GenFv tool when creating FV images. The following example is what this generated, FV-style, INF file looks like:

```
[options]
EFI_BLOCK_SIZE = 0x10000

[attributes]
EFI_ERASE_POLARITY   =  1
EFI_WRITE_ENABLED_CAP = TRUE
EFI_READ_ENABLED_CAP = TRUE
EFI_READ_LOCK_STATUS = TRUE
EFI_WRITE_STATUS = TRUE
EFI_READ_DISABLED_CAP = TRUE
EFI_WRITE_LOCK_STATUS = TRUE
EFI_LOCK_CAP = TRUE
EFI_LOCK_STATUS = TRUE
EFI_ERASE_POLARITY = 1
EFI_MEMORY_MAPPED = TRUE
EFI_READ_LOCK_CAP = TRUE
EFI_WRITE_DISABLED_CAP = TRUE
EFI_READ_STATUS = TRUE
EFI_WRITE_LOCK_CAP = TRUE
EFI_STICKY_WRITE = TRUE
EFI_FVB2_ALIGNMENT_16 = TRUE

[files]
EFI_FILE_NAME =
C:\Combo\R9\LakeportX64Dev\Build\LakeportX64Pkg\DEBUG_MYTOOLS\FV\Ffs\EDBEDF47-
6EA3-4512-83C1-70F4769D4BDE\EDBEDF47-6EA3-4512-83C1-70F4769D4BDE.ffs
```

9.5 Create the FD image file(s)

The whole FD image is described by a list of `Regions` which correspond to the locations of different areas within the hardware flash device. Currently most flash devices have a variable number of blocks, all of identical size. When "burning" an image into one of these devices, only whole blocks can be burned into the device at any one time. This puts a constraint that all layout regions of the FD image must start on a block boundary. To accommodate future flash parts that have variable block sizes, the layout is described by the offset from the `BaseAddress` and the size of the section that is being described. Since completely filling a block is not probable, part of the last block of a region can be left empty. To ensure that no extraneous information is left in a partial block, the block should be erased prior to burning it into the device. Multiple devices with non-volatile memory are treated as a single device with contiguous memory space.

Regions must be defined in ascending order and may not overlap.

Each layout region starts with a eight digit hex offset (leading "`0x`" required) followed by the pipe "|" character, followed by the size of the region, also in hex with the leading "`0x`" characters.

The format for an FD Layout Region is:

```
Offset|Size
[TokenSpaceGuidCName.PcdOffsetCName|TokenSpaceGuidCName.PcdSizeCName]?
[RegionType]?
```

Setting the optional PCD names in this fashion is shortcut. The two regions listed below are identical, with the first example using the shortcut, and the second using the long method:

```
0x000000|0x0C0000
gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress|gEfiMyTokenSpaceGuid.PcdFlashFvMa
inSize
FV = FvMain

0x000000|0x0C0000
SET gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress = 0x000000
SET gEfiMyTokenSpaceGuid.PcdFlashFvMainSize = 0x0C0000
FV = FvMain
```

The shortcut method is preferred, as the user does not need to maintain the values in two different locations.

The optional **RegionType**, if specified, must be one of the following **FV**, **DATA**, or **FILE**. or no **RegionType** at all. Not specifying the **RegionType** implies that the region starting at the "*Offset*", of length "*Size*" should not be touched. This unspecified region type is typically used for event logs that are persistent between system resets, and modified via some other mechanism (and SMM Event Log module, for example).

EDK II FDF does not use the concept of sub-regions, which existed in EDK FDF files.

9.5.1 FV Region Type

The **FV RegionType** is used as a pointer to either one of the unique FV names that are defined in the **[FV]** section or a file specified in an **[FV]** section's **CREATE_FILE = statement**. Both of these are files that contains a binary FV as defined by the PI 1.2 specification. The format for the **FV RegionType** is one of the following:

```
FV = $(UiFvName)
```

Or

```
FV = $(Filename)
```

Where **$(Filename)** is a filename defined by the **[FV]** section's **CREATE_FILE = <Filename>** statement.

The following is an example of FV region type.

```
0x000000|0x0C0000
gEfiMyTokenSpaceGuid.PcdFlashFvMainBaseAddress|gEfiMyTokenSpaceGuid.PcdFlashFvMa
inSize
FV = FvMain
```

9.5.2 DATA Region Type

The DATA **RegionType** is a region that contains is a hex value or an array of hex values. This data that will be loaded into the flash device, starting at the first location pointed to by the *Offset* value. The format of the DATA **RegionType** is:

```
DATA = { <Hex Byte Data Structure> }
```

The following is an example of a DATA region type.

```
0x0CA000|0x002000
gEfiMyTokenSpaceGuid.PcdFlashNvStorageBase|gEfiMyTokenSpaceGuid.PcdFlashNvStorag
eSize
DATA = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x8D, 0x2B, 0xF1, 0xFF, 0x96, 0x76, 0x8B, 0x4C
}
```

This data may need to be modified based on content of the region. In order for EFI modules to access these regions, a customized region header may be required. Tools

for creating custom header information is beyond the scope of the standard build.

9.5.3 FILE Region Type

The **FILE RegionType** is a pointer to a binary file that will be loaded into the flash device, starting at the first location pointed to by the *Offset* value. The format of the **FILE RegionType** is:

```
FILE = $(FILE_DIR)/Filename.bin
```

The following is an example of the **FILE RegionType**.

```
0x0CC000|0x002000
gEfiCpuTokenSpaceGuid.PcdCpuMicrocodePatchAddress|gEfiCpuTokenSpaceGuid.PcdCpuMi
crocodePatchSize
FILE = FV/Microcode.bin
```

10 ImageGen Stage - Other Images

This chapter describes the processing of the EFI files generated by the $(MAKE) Stage into images such as Applications or images used by PCI Option ROMs and/or Update Capsules. Creating images that do not go into a flash part directly such as stand-alone Applications and PCI Option ROM images, do not need an FDF file. This also applies to binary driver images that are to be used for a binary distribution - the files for these images are created during the $(MAKE) stage.

10.1 EFI PCI OptionROM Images

To generate the EFI PCI OptionROM, the EFI PE32 files and optionally the legacy OptROM (from a separate tool) are needed.

The EfiRom tool is used on the PE32 and optionally the legacy OptionROM binary images. The tool will checkthe header of each file to determine the type.

- If the input file(s) are EFI PE32 image,
  - fill in EFI PCI OptROM header and PCI data structure in the output EFI PCI OptionROM image
  - then copy the input EFI PE32 file content to the output EFI PCI OptionROM image to create the EFI PCI OptionROM image.
- If the input file(s) are legacy OptROM binary image,
  - fill in EFI PCI OptROM header in the output EFI PCI OptionROM image
  - then copy the input file content to the output EFI PCI OptionROM image to create the EFI PCI OptionROM image.

The final image is placed in the FV folder of the build directory.

10.2 UEFI Applications

If a developer wants to generate only UEFI applications, verify that no FDF file is specified in the DSC file. This prevents the GenFds tool from being called after all of the modules have been built by the $(MAKE) stage. The UEFI application files (.efi files) built from application modules are put in the following directory:

`$(OUTPUT_DIRECTORY)/$(PLATFORM_NAME)/<BuildTarget>_<ToolChainTag>/$(ARCH)`

10.3 Capsules

This section describes the processing of the EFI files generated by the $(MAKE) Stage into Update Capsules. Capsule images contain a Fv Image or a FFS file to be updated.

The `[Capsule]` section in FDF file is parsed to get:

- Capsule Header information, including: Capsule guid, flags and header size.
- Capsule content may be either a Fv Image or a FFS file.

A Fv Image may be specified using any FV section described in this FDF file. It will be generated same using the process described in Section 9.4. Additionally, an existing FV file created as part

of an FD image may be used. These FV files can be directly integrated into a Capsule. Raw data (non-FFS files) can be included in a FV file, using EFI_FV_FILETYPE_RAW.

The FFS file contains EFI section files (see Table 2 for a list of EFI_SECTION types. All files generated by the $(MAKE) stage, will have the output located in a build directory, either at the top of: $(OUTPUT_DIRECTORY)/$(PLATFORM_NAME)/<BuildTarget>_<ToolChainTag>/$(ARCH) or a sub-directory created that replicates the INF file path. All EFI section files and encapsulated section files are created based on their description in FDF file. For a binary or raw file type, the raw data can be any binary file. One FV image or one FD image described in FV section or FD sections of the FDF file may also be treated as RAW data. The process of creating a FV image is described in Section 9.4, the process of creating a FD image is described in Section 9.5.

The $(MAKE) stage creates EFI files. During the ImageGen stage, GenFds will create the required FFS files and FV images based on [Capsule] description in the FDF file. Finally, the capsule header will be prefixed to the capsule data to construct the complete capsule. The overview of the Capsule creation process is shown in Figure 23:

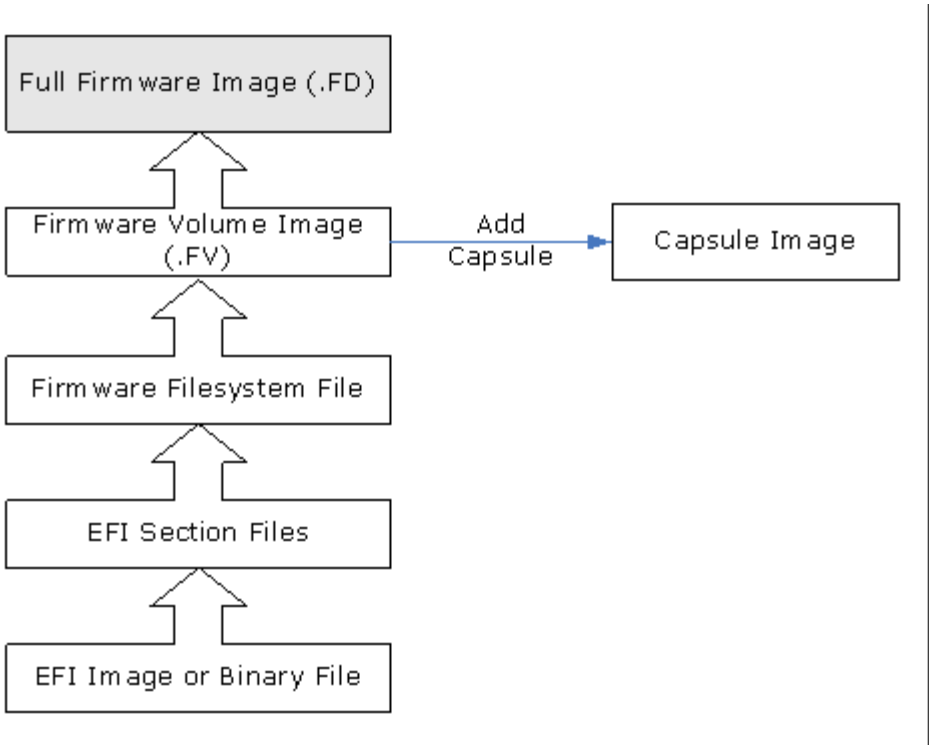


Figure 23. Capsule Creation Process.

11 Build Changes and Customizations

This chapter deals with customizing a build, including options and settings for debugging, using custom tools as well as how to customize EDK component builds

11.1 Building for Debug

The build tool defaults support two building targets: DEBUG and RELEASE. This section describes how to enable DEBUG target when building and how to setup compiler flags used for DEBUG.

There are three ways provided in build tool to define which target should be used in the building process.

- Situation A: Setup by overriding file "target.txt"
  - — After executed command "edksetup" in workspace environment, there will be a file named "target.txt" under $(WORKSPACE)/Conf.
  - — Users can edit this file and change the value of item "TARGET".
  - — A specific example of this is "TARGET = DEBUG", which sets the current building method.
  - — In this example, the default value of the "TARGET" is set to "DEBUG".
- Situation B: Use a parameter "-b BUILDTARGET" when executing building command
  - — Users can type a command with the format "build -b BUILDTARGET"to specify the target used in current building.
  - — A specific example of this command is "build -b DEBUG".
  - — In this example, the value set in the file "target.txt" will be ignored.
- Situation C: Setup in the DSC file of a platform
  - — When the BUILDTARGET is not specified in the command line or in the file "target.txt", the build tool will attempt to build all valid targets specified in the DSC file.
  - — This contrasts with situations A and B, where only the targets specified as valid in the DSC file can be used.

11.1.1 Debugging Files

For a debugging build, the files created will be saved to $(WORKSPACE)/ $(OUTPUT_DIRECTORY)/$(BUILDTARGET)_$(TOOL_CHAIN_TAG)/$(ARCH)/. For each single module, the temporary files created in DEBUG building process will be saved to $(WORKSPACE)/$(OUTPUT_DIRECTORY)/$(BUILDTARGET)_$(TOOL_CHAIN_TAG)/ $(ARCH)/$(PACKAGE_NAME)/$(MODULE_NAME)/DEBUG/ and $(WORKSPACE)/ $(OUTPUT_DIRECTORY)/$(BUILDTARGET)_$(TOOL_CHAIN_TAG)/$(ARCH)/ $(PACKAGE_NAME)/$(MODULE_NAME)/OUTPUT/, so such as .map, .pdb and other DEBUG files can be found in these two directories.

User can also define a specific directory to save DEBUG files. A detailed example is given in the next subsection.

11.1.2 Debugging Options

Build tool supports customized DEBUG flags in the <BuildOptions>section of the DSC file, the INF file and "tools_def.txt". The highest priority for a same complier flag is the one defined in INF file, the medium is that in DSC file and the lowest is the one in "tools_def.txt".

For example, to generate the .cod files for the .obj files of a platform, user can add one line such as *_MYTOOLS_*_CC_FLAGS = /FAcs /FA $(OUTPUT_DIR)\ in section [BUILD_OPTIONS] of DSC file. This option tells build tool to generate a .cod file for each .obj file and put them to $(OUTPUT_DIR)

For only generating the .cod files for one single module, one way is to add the option in section [BUILD_OPTIONS] of the module's INF file; anther way is to add the option to DSC file's <BuildOptions> for the INF file like below:

```
MdeModulePkg/Universal/PCD/Pei/Pcd.inf {
  <BuildOptions>
    *_MYTOOLS_*_CC_FLAGS = /FAcs /FA $(OUTPUT_DIR)\
}
```

Please refer to 5.2 tools.def.txt and Appendix B tools_def.txt for more detailed information about build options.

11.1.3 Advanced Debugging

For generating disassembly (.cod) files for debugging, the following is one way to setup dumpbin -disasm for individual modules as well as using it for every .efi file generated.

To generate the disasm for the efi files, the user can add two definitions in "tools_def.txt":

```
DEBUG_MYTOOLS_IA32_DISASM_PATH  = DEF(VS2005TEAMSUITE_BIN)\dumpbin.exe
DEBUG_MYTOOLS_IA32_DISASM_FLAGS = -dump -disasm -out:$(DEST_DIR)
```

Then user can use build option -y, --define with a reserved MACRO name: DISASM to start building. The build tool automatically detects if a DISASM tool defined in the Tagname of ToolChain, then after ever link command that generates an EFI file, the tool will run the DISASM tool (with the flags) against the EFI file. In the example, the output file will be next to the EFI file based on the FLAGS entry, -out:$(DEST_DIR) which is the same location as the .efi file.

11.2 Adding Custom Compression Tools

This section covers how to add a customized compress tool, such as TianoCompress tool.

First, one specific GUID is assigned to the added tool, which can be used to specify this tool and its compressed data. Then the tool path and guid needs to be added into tools_def.txt file, for example TianoCompress tool used for all tool chains, Target and Archs can be added like:

```
*_*_*_TIANOCOMPRESS_PATH  = DEF(TOOL_PATH)\TianoCompress.exe
*_*_*_TIANOCOMPRESS_GUID    = A31280AD-481E-41B6-95E8-127F4C984779
```

Next, "$(TOOLNAME)" can be specified in build_rule.txt file to call this tool. And, its guid value is supported in FDF file to call this tool, which is used to create the EFI guided section data. For TianoCompress tool, "$(TIANOCOMPRESS)" is used in build_rule.txt file, A31280AD-481E-41B6-95E8-127F4C984779 is used in FDF file.

NT32.fdf file uses TianoCompress tool to create the guided data like:

```
[Rule.Common.PEIM.TIANOCOMPRESSED]
  FILE PEIM = $(NAMED_GUID) DEBUG_MYTOOLS_IA32 {
    PEI_DEPEX PEI_DEPEX Optional         |.depex
    GUIDED A31280AD-481E-41B6-95E8-127F4C984779 {
      PE32      PE32                     |.efi
      UI        STRING="$(MODULE_NAME)" Optional
      VERSION   STRING="$(INF_VERSION)" Optional BUILD_NUM=$(BUILD_NUMBER)
    }
  }
```

11.3 Using Custom Build Tools

This section introduces how to use the custom tools in EDKII build system.

The custom tools can be classified to two types. One is used to create the EFI guided section data, which must have its matched GUID value, such as the custom compression tool introduced in the last section. Another is only used to process files, which may not require its GUID, such as ASL compiler. This section focuses on the later one.

First, the custom tool path and name needs to be added into tools_def.txt file. For ASL compiler, it can be added like:

```
*_*_*_ASL_PATH          = DEF(ASL_PATH)\iasl.exe
```

Then, "$(TOOLNAME)" is specified in build_rule.txt file to call this tool. For ASL compiler, it can be used to process ASL source file. The rule to process ASL file is added in build_rule.txt like:

```
[Build.Acpi-Source-Language-File]
    <InputFile>
        ?.asl, ?.Asl, ?.ASL
    <OutputFile>
        $(OUTPUT_DIR)(+)${s_base}.aml
    <Command.MSFT, Command.INTEL>
        "$(PP)" $(APP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
        "$(ASL)" -p ${dst} ${d_path}(+)${s_base}.i
```

11.4 Customizing Compilation for a Component

There are several mechanisms for customizing the build for a firmware component. These include:

- Creating a new Platform (DSC) file from an existing platform.
- Creating a custom INF file for individual components or modules.
- Using MACRO definitions with control statements (!ifxxx) in the DSC and FDF files.
- Customizing the INF build options in the DSC file.

11.5 Changing Files in a Library [EDK I Only]

Library INF files are shared among different platforms. However, not all platforms require all the same source files. To customize the library INF files for different platforms, simply define $(PLATFORM), either on the command line, or in the DSC file,

and then make customizations in the [sources.$(PROCESSOR).$(PLATFORM)] section of the library INF file.

An alternative to this method is to simply create a new INF file for the library, and then use it in place of the existing library INF file.

12 Reports

This section introduces the build report generation tool functionality and its output report format. It describes the external behaviors of the tool, i.e. the accepted command line options and the detailed output report format.

12.1 Build Report Generation (BRG) Tool

Report Generation tool is part of build process to report the following platform information after platform build ends successfully.

- "PCD INFO (Complete platform configuration database information)
- "LIBRARY INFO (Library class and instance mapping, constructor/destructor information)
- "DEPEX INFO (Module dependency information)
- "BUILD INFO (Module build tool chain tag, specific compiler or linker options)
- "FLASH INFO (Module firmware device and firmware volume information)
- "PREDICTION INFO (The predicted dispatch order of modules (PEIMs / drivers) and their notification invoking sequence; Also the predicted addresses of module image loading, entry point and notification functions.)

The information in the reports listed above is useful for platform integrators to diagnose the platform issues in an efficient way. Integrators must specify which reports to include in the report file.

12.2 Sample Launch Steps: NT32 platform

BRG functionality is switched on by "-y" or "-Y" option from build command. The following steps output the build report for NT32 platform:

1. Check out edk2 packages from https://edk2.tianocore.org/svn/edk2/trunk/edk2 to c:\edk2 directory
2. Run Cmd.exe and enter c:\edk2
3. Run "edksetup.bat --nt32"
4. Run "build.exe -p Nt32Pkg\Nt32Pkg.dsc -y ReportFile.txt -Y ALL”
   a -y: This option specifies the output file name for build report.
   b -Y: This option specifies flags that control the type of build report. It must be from set of ALL, PCD, LIBRARY, DEPEX, BUILD_FLAGS, FLASH and PREDICTION. To specify more than one flag, repeat the option on the command line. Example of usage:
   On the command line “-y report_filename.ext -Y PCD -Y FLASH -Y DEPEX”.

12.3 Output

The output is in raw text file encoded in ASCII character set so that it can be portable to all OS environments. The text file is supposed to be organized in a logical way for

human readability and QA team's validation.

12.3.1 Layout

The whole layout of BRG tool report:

```
|---- Platform summary
    |----- Global PCD section
    |----- FD section*
        |---- FD Region sub-section*
    |---- Module section*
        |---- Basic Information summary
        |---- PCD sub-section
        |---- Library sub-section
        |---- DEPEX sub-section
        |---- Build_flags sub-section
        |---- Notification sub-section
```

Note: *Items marked with * can occur more than once in one parent instance.*

12.3.2 Section and Sub-section Format

The output report of BRG is divided into platform and module part. Each part may further consist of sections and sub-sections with the following rules:

1. Each section starts with marker `>==============================<`
2. Each section ends with marker `<==============================>`
3. There must be a section header after each section start marker.
4. There must a separator `==========================` to separate the section header and contents if the section has non-empty contents.
5. The section contents can further be divided into one-level sub-sections.
6. Each sub-section starts with marker `>----------------------------------------<`
7. Each sub-section ends with marker `<---------------------------------------->`
8. There must be a sub-section header after each section start marker.
9. There must a separator `----------------------------------------` to separate the section header and contents if the section has non-empty contents.
10. In general, each line in section should not exceed 120 characters.

Example:

```
Platform Name: Nt32
Platform DSC Path: c:\edk2\Nt32Pkg\Nt32Pkg.dsc
Architectures: IA32
Tool Chain: MYTOOLS
Target: DEBUG
Build Environment: Windows-XP-5.1.2600
Build Duration: 05:12
Report Contents:LIBRARY, FLASH, PREDICTION
>============================================================================<
Firmware Device (FD)
FD Name: Nt32
Base Address: 0
Size: 0x2a0000(2688KB)
==============================================================================
<----------------------------------------------------------------------------->
FD Region
Type: FV
Base Address: 0
Size: 0x280000 (2560K)
FV Name: FvRecovery
Occupied Size: 0x221F60 (2183K)
Free Size: 0x5E0A0 (376K)
------------------------------------------------------------------------------
..(List of Module in FvRecovery)
>----------------------------------------------------------------------------<
..(List of other FD region sub-section)
>============================================================================<
```

The following sections describe these reports and sub-sections in detail.

12.4 Platform Summary

Platform summary displays at the beginning of the output report, including the following items:

- Platform Name : %Platform UI name: 'PLATFORM_NAME' in DSC [Defines] section%
- Platform DSC Path: %Path of platform DSC file%
- Architectures : %List string of all architectures used in build%
- Tool Chain : %Tool chain string%
- Target : %Target String"
- Build Environment : %Environment string reported by Python%
- Build Duration : %Build duration time string%
- Report Content : %List of flags the control the report content%

Example:

```
Platform Name: Nt32
Platform DSC Path: c:\edk2\Nt32Pkg\Nt32Pkg.dsc
Architectures: IA32
Tool Chain: MYTOOLS
Target: DEBUG
Build Environment: Windows-XP-5.1.2600
Build Duration: 05:12
Report Contents: LIBRARY, FLASH, PREDICTION
```

Note: *Platform Summary always presents and appears at the beginning of report.*

12.5 Global PCD Section

This section contains the information for all PCDs whose values are the same for all modules in a platform. The content of global PCD sub-section is grouped by token space:

```
[gEfiNt32PkgTokenSpaceGuid]
…
…
[gEfiMdeModulePkgTokenSpaceGuid]
…
…
…
```

Each global PCD item contains one or more lines:

12.5.1 Required line

The first line is required:

```
[*P|*F| ] <PcdCName>: <PcdType> (<DatumType>) = <PcdValue>
```

- *P means the Pcd's value is override in DSC file
- *F means the PCD's value is override in FDF file.
- If no *P or *F, mean the PCD's value comes from DEC file.

Example:

```
*P PcdWinNtFirmwareVolume :  FIXED    (VOID*) = L"..\\Fv\\Nt32.fd"
```

12.5.2 Optional lines

12.5.2.1 First optional line

- if <PcdType> is DYN-HII

```
<VariableGuid>:<VariableName>:<Offset>
```

Example:

```
*P PcdGMchDvmtTotalSize : DYN-HII (UINT8) = 0
                          gSysConfigGuid: L"Setup": 0xAA
```

- if <PcdType> is DYN-VPD

```
<Offset relative to VPD base address>
```

Example:

```
*F PcdVpdSample : DYN-VPD (UINT32) = 1
                  0x0001FFF
```

12.5.2.2 Second optional line

The second optional line is present if the value from the DEC was overridden. It is formatted as follows:

```
DEC DEFAULT = <Value in DEC>
```

Example:

```
*P PcdWinNtFirmwareFdSize    :  FIXED    (UINT32) = 0x2a0000
                                     DEC DEFAULT = 0x0
```

12.5.2.3 Additional optional lines

Additional lines are optional and show module level overrides. Each module with an override is listed with the value of the override.

- *M means the PCD's value is module level override.

Each line is formatted as follows:

```
*M Inf Filename
```

Example:

```
*P PcdDebugPrintErrorLevel    :  PATCH    (UINT32) = 0x80000042
                                     DEC DEFAULT = 0x80000000
*M Tcp4Dxe.inf                                   = 0x80000000
```

Note: *Global PCD section presents when PCD is specified in -Y option.*

12.6 FD Section

This section contains platform flash device information and its layout.

12.6.1 FD Section Header

Given that a platform may have multi-Firmware device, this section may appear more than once in the output report. The section header lists the name of FD and its base address and size. The contents of the section consist of one or more FD region sub-section.

- FD Name : %FD UI name: FD file base name%
- Base Address: %Base address for the FD image%
- Size : %Size of the FD image%

Example:

```
>=============================================================================<
Firmware Device (FD)
FD Name: Nt32
Base Address: 0
Size: 0x2a0000(2688KB)
==============================================================================
... (one or more FD Region Sub-section)
<=============================================================================>
```

12.6.2 FD Region Sub-section

This sub-section contains FD region information of platform flash device. If the region is a firmware volume, it lists the set of modules and its space information; otherwise, it only lists its region name, base address and size in its sub-section header.

- Region Type : %The type of the FD region (FV, Data, File or None)%
- Base Address: %Base address for the FD region%
- Size : %Size of the FD region%
- FV Name*: %FV name and occupation percentage%
- Occupied Size*: %The occupied size of the FV%
- Free Size*: %The free size of the FV%

The contents of FD region sub-section contain the list:

```
(Offset, Module)*: %The list offset and module INF file path in the FV%
```

The items marked with * are only available when the region type is FV.

Example1:

```
<-----------------------------------------------------------------------------><
FD Region
Type: FV
Base Address: 0
Size: 0x280000 (2560K)
FV Name: FvRecovery
Occupied Size: 0x221F60 (2183K)
Free Size: 0x5E0A0 (376K)
Offset   Module
-------------------------------------------------------------------------------
0x000048 PeiAprioriFileName
0x000078 Apriori
0x000FE8 PeiCore (c:\edk2\MdeModulePkg\Core\Pei\PeiCore.inf)
0x011FE8 PcdPeim (c:\edk2\MdeModulePkg\Universal\Pcd\Pei\PcdPei.inf)
...(More list of offset and modules)
>-----------------------------------------------------------------------------<
```

Example2:

```
<-------------------------------------------------------------------------->
FD Region
Type: DATA
Base Address: 0x280000
Size: 0xc000 (48K)
>--------------------------------------------------------------------------<
...(More list of FD regions)
```

Note: *The whole FD section presents when FLASH (or ALL) is specified in -Y option.*

12.7 Module Section

Module section lists all modules involved in the platform build. If PREDICTION option is specified in -Y option, the module sections are sorted according to their PEI or DXE dispatch order; otherwise the module sections are listed according to their DSC position.

12.7.1 Module Section Summary

This sub-section lists the module basic information: Module name: file Guid; module size; module build time stamp; driver type; Specification version.

- Module Name : %Module UI name: 'MODULE_NAME' in INF [Defines] section%
- Module INF Path: %Path of Module INF file%
- File GUID: %Module GUID: 'FILE_GUID' in INF [Defines] section%
- Size: %Module PE32 image size%
- Build time stamp: %The time stamp in module PE32 image%
- Driver Type: %The driver file type%
- UEFI Specification Version: %The UEFI specification version the module conform to%
- PI Specification Version: %The PI specification version the module conform to%
- Dispatch Order*: The predicted dispatch order of this module
- Image Address*: %The predicted image loading address%
- Entry Point Address*: %The predicted entry point address%

The items marked with * exist when PREDICTION (or ALL) is specified in -Y option.

Example1:

```
>============================================================================<
Module Summary
Module Name: SmbiosDxe
Module INF Path: c:\edk2\MdeModule\Universal\SmbiosDxe\SmbiosDxe.inf
File GUID: F9D88642-0737-49BC-81B5-6889CD57D9EA
Size: 0x1E2C (7.54K)
Time Stamp: 2009-12-08 13:30:00.0
Driver Type: 0x7 (DRIVER)
UEFI Specification Version: 2.1
PI Specification Version: 1.0
Dispatch Order: 11 (DXE Dispatcher)
Image Address: TOM - 0xf8a000
Entry Point Address: TOM - 0xf89a60
============================================================================
... (Module Section Details for SmbiosDxe)
<============================================================================>
```

Example2:

```
>============================================================================<
Module Summary
Module Name: EbcDxe
Module INF Path: c:\edk2\MdeModule\Universal\EbcDxe\EbcDxe.inf
File GUID: 13AC6DD0-73D0-11D4-B06B-00AA00BD6DE7
Size: 0x35B8 (13.43K)
Time Stamp: 2009-12-08 13:30:00.0
Driver Type: 0x7 (DRIVER)
UEFI Specification Version: 2.1
============================================================================
... (Module Section Details for EbcDxe)
<============================================================================>
```

12.7.2 Library Sub-section

This sub-section holds the information for all libraries used in this module. If it is an EDKII style module, it further lists its correspondent library class, library constructor and destructor name if they exist. The library instances are sorted by the order of their constructor calling sequence and the reverse order of their destructor calling sequence.

- Library INF Path: %Path of library INF file%
- Class*: %The library class name of the library instance%
- C*: %The library constructor if it exists%
- D*: %The library destructor if it exists%

The items marked with * are only available when the module is an EDKII style module and they must list in the next line immediately after library INF path.

Example1:

```
>------------------------------------------------------------------------------<
Library
------------------------------------------------------------------------------
c:\edk2\MdePkg\Library\UefiDevicePathLib\UefiDevicePathLib.inf
{DevicePathLib}
c:\edk2\MdePkg\Library\BaseLib\BaseLib.inf
{BaseLib}
c:\edk2\MdePkg\Library\BaseMemoryLib\BaseMemoryLib.inf
{BaseMemoryLib}
c:\edk2\MdePkg\Library\UefiMemoryAllocationLib\UefiMemoryAllocationLib.inf
{MemoryAllocationLib}
c:\edk2\MdePkg\Library\UefiBootServicesTableLib\UefiBootServicesTableLib.inf
{UefiBootServicesTableLib: C = UefiBootServicesTableLibConstructor}
c:\edk2\MdePkg\Library\DxePcdLib\DxePcdLib.inf
{PcdLib: C = PcdLibConstructor}
c:\edk2\MdePkg\Library\UefiRuntimeServicesTableLib\UefiRuntimeServicesTableLib.i
nf
{UefiRuntimeServicesTableLib: C = UefiRuntimeServicesLibConstructor}
c:\edk2\MdePkg\Library\BaseIoLibIntrinsic\BaseIoLibIntrinsic.inf
{IoLib}
c:\edk2\MdePkg\Library\BasePciCf8Lib\BasePciCf8Lib.inf
{PciCf8Lib}
c:\edk2\MdePkg\Library\BasePciLibCf8\BasePciLibCf8.inf
{PciLib}
c:\edk2\MdePkg\Library\BasePrintLib\BasePrintLib.inf
{PrintLib}
c:\edk2\Ich9Pkg\Library\IntelIchAcpiTimerLib\IntelIchAcpiTimerLib.inf
{TimerLib: C = IntelAcpiTimerLibConstructor}
c:\edk2\MdePkg\Library\UefiLib\UefiLib.inf
{UefiLib}
c:\edk2\MdePkg\Library\BaseSynchronizationLib\BaseSynchronizationLib.inf
{SynchronizationLib}
c:\edk2\MdePkg\Library\DxeHobLib\DxeHobLib.inf
{HobLib: C = DxeHobLibConstructor}
c:\edk2\MdePkg\Library\UefiDriverEntryPoint\UefiDriverEntryPoint.inf
{UefiDriverEntryPoint}
c:\edk2\MdePkg\Library\UefiRuntimeLib\UefiRuntimeLib.inf
{UefiRuntimeLib: C = UefiRuntimeLibConstructor D = UefiRuntimeLibDestructor}
<------------------------------------------------------------------------------>
```

Example2:

```
>----------------------------------------------------------------------------<
Library
----------------------------------------------------------------------------
c:\edk2\R8BearLakePkg\Guid\GuidLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Guid\EdkGuidLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Protocol\EdkProtocolLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Library\RuntimeDxe\EfiRuntimeLib\EfiRunti
meLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Core\Dxe\ArchProtocol\ArchProtocolLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Library\CompilerStub\CompilerStubLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Guid\EdkGuidLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Framework\Protocol\EdkFrameworkProtocolLi
b.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Efi\Guid\EfiGuidLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Efi\Protocol\EfiProtocolLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Library\EfiCommonLib\EfiCommonLib.inf
c:\edk2\EdkCompatibilityPkg\Foundation\Framework\Guid\EdkFrameworkGuidLib.inf
<----------------------------------------------------------------------------->
```

Note: *This sub-section presents when LIBRARY (or ALL) is specified in -Y option.*

12.7.3 PCD Sub-section

This sub-section holds the information for all PCDs used in this module. The content of module PCD sub-section is divided by token space such as:

```
[gEfiNt32PkgTokenSpaceGuid]
…
…
[gEfiMdeModulePkgTokenSpaceGuid]
…
…
…
```

Each item of module PCD maybe contains three lines:

1. The first line is a mandatory lines as:

```
[*P|*F| ] <PcdCName>: <PcdType> (<DatumType>) = <PcdValue>
```

*P means the Pcd's value is override in DSC file

*F means the PCD's value is override in FDF file.

If no *P or *F, mean the PCD's value comes from DEC file.

For example:

```
*P PcdWinNtFirmwareVolume :  FIXED    (VOID*) = L"..\\Fv\\Nt32.fd"
```

2. The second line is the optional line
   a if <PcdType> is DYN-HII

```
<VariableGuid>:<VariableName>:<Offset>
```

For example:

```
*P PcdGMchDvmtTotalSize : DYN-HII (UINT8) = 0
                          gSysConfigGuid: L"Setup": 0xAA
```

   b if <PcdType> is DYN-VPD

```
<Offset relative to VPD base address>
```

For example:

```
*F PcdVpdSample : DYN-VPD (UINT32) = 1
                  0x0001FFF
```

3. The third line is the optional line if <PcdValue> in DSC file is not equal to the value in DEC files.

```
DEC DEFAULT = <Value in DEC>
```

For example:

```
*P PcdWinNtFirmwareFdSize    :  FIXED    (UINT32) = 0x2a0000
                                DEC DEFAULT = 0x0
```

Note: *This sub-section presents when PCD (or ALL) is specified in -Y option.*

12.7.4 DEPEX Sub-section

This sub-section holds module dependency expression (DEPEX) information. The sub-section header holds the module finial dependency expression. If the module is an EDKII style module and it inherits dependency from one of its library instance, it lists the inherited library dependency expression in the sub-section contents.

Example1:

```
>---------------------------------------------------------------------------<
Dependency Expression (DEPEX) from INF
(gEfiFirmwareVolumeBlockProtocolGuid) AND (gEfiPcdProtocolGuid)
-----------------------------------------------------------------------------
Module: gEfiFirmwareVolumeBlockProtocolGuid
From Lib: gEfiPcdProtocolGuid c:\edk2\MdePkg\Library\DxePcdLib\DxePcdLib.inf
<---------------------------------------------------------------------------->
```

Example2:

```
>---------------------------------------------------------------------------<
Dependency Expression (DEPEX) from DXS
EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_GUID AND EFI_VARIABLE_ARCH_PROTOCOL_GUID AND
EFI_VARIABLE_WRITE_ARCH_PROTOCOL_GUID AND EFI_METRONOME_ARCH_PROTOCOL_GUID AND
EFI_RUNTIME_ARCH_PROTOCOL_GUID AND EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_GUID AND
EFI_HII_DATABASE_PROTOCOL_GUID
<---------------------------------------------------------------------------->
```

Note: *This sub-section presents when DEPEX (or ALL) is specified in -Y option.*

12.7.5 Build Flags Sub-section

This sub-section holds module build flags information. The sub-section header holds the module tool chain tag and the sub-section contents list all related build flags.

Example:

```
>---------------------------------------------------------------------------<
Build Flags
Tool Chain Tag: MYTOOLS
-----------------------------------------------------------------------------
CC_FLAGS = /nologo /c /WX /GS- /W4 /Gs8192 /Gy /D UNICODE /O1ib2 /GL /FIAutoGen.h
/EHs-c- /GR- /GF /Zi /Gm /D EFI_SPECIFICATION_VERSION=0x0002000A /D
TIANO_RELEASE_VERSION=0x00080006 /Od
DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /OPT:ICF=10 /MAP /
ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /DLL /
```

```
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /DEBUG
/PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb
<-------------------------------------------------------------------------->
```

Note: *This sub-section presents when BUILD_FLAGS (or ALL) is specified in -Y option.*

12.7.6 Notification Sub-section

This sub-section contains module notification functions information. This sub-section does not exist if the module does not have notification functions. All the notification functions are listed with the following triple line by line:

- (Address, Order, Name):%The predicted address, order and name of the notification function%

Example:

```
>-------------------------------------------------------------------------<
Notification Functions Information
Address    Seq Name
-------------------------------------------------------------------------
0xfffff8a38 3  MemoryDiscoveredPpiNotifyCallback
0x3ef8a8a38 6  EndOfPeiSignalPpiNotifyCallback
<-------------------------------------------------------------------------->
```

Note: *This sub-section presents when PREDICTION (or ALL) is specified in -Y option.*

Appendix A
Variables

One of the core concepts of this utility is the notion of symbols. Use of symbols follows the makefile convention of enclosing within $(), for example $(EFI_SOURCE). As the utility processes files during execution, it will often perform parsing of variable assignments. These variables can then be referenced in other sections of the DSC file. Variable assignments will be saved internally in either a local or global symbol table. The local symbol table is purged following processing of individual Platform (DSC) files. Global symbol values persist throughout execution of the utility. Local symbol values take precedent over global symbols. The following table describes the symbols generated internally by the utility. They can be overridden either on the command line, in the DSC file, or in individual INF files. The G/L column indicates whether the symbol is typically a global or a local symbol.

Variable descriptions follow in Table 27.

Note: *This table does not list required system environment variables or optional system environment variable.*

Table 27. Variable Descriptions

Variable Name	G/L	Description
BIN_DIR	L	Specifies the directory where final component binaries are deposited during build. Typically $(BUILD_DIR)\$(PROCESSOR)
BUILD_DIR	G	Defines the build tip directory for the current platform. For example, this may be $(EFI_SOURCE)\Platform\Anacortes_870.
BUILD_TYPE	L	If defined, then the utility will copy the [build.$(PROCESSOR).$(BUILD_TYPE)] section from the DSC file to the component's makefile. If not specified, then the [build.$(PROCESSOR).$(COMPONENT_TYPE) section will be used to emit command to build the component.
DEST_DIR	L	For a component, defines the directory (typically under BUILD_DIR) where the component object files are to be built.
DSC_FILENAME	G	Name of the DSC file as specified on the command line. Can be used for dependencies in the makefiles.
EDK_SOURCE	G	Defines the root directory of the local EFI source tree, for example C:\EFI2.0. If not defined as an environmental variable when the tool is invoked, the utility will attempt to determine a reasonable value based on the current working directory.
FILE	L	As the utility processes each source file in the Platform (DSC) file, this symbol gets assigned the name of the file, less the file extension.
FV_EXT	L	Common component type (BS driver, application, etc) have predefined file name extensions assigned (.dxe, .app, etc). If there is a deviation from the convention, or a new (unknown to the utility) component type is being built, then FV_EXT may need to be defined for the component so the utility knows the result file name extension. This information is necessary to generate dependencies in makefile.out.
INF_FILENAME	L	Name of the INF file for a given component. Can be used for dependencies in the makefiles.
LIB_DIR	L	Specifies the directory where EFI libraries are deposited after building. Typically $(BUILD_DIR)\$(PROCESSOR)
MAKEFILE_NAME	L	Name of the output makefile for the component. Default is "makefile". This value can be overridden to support building different variations of a component in the same DEST_DIR directory.
OUT_DIR	L	Unused, but typically $(BUILD_DIR)\$(PROCESSOR)
PACKAGE	L/G	If defined, then the utility will create a package file named $(DEST_DIR)\$(BASE_NAME).pkg, and copy, with macro expansion, the [package.$(COMPONENT_TYPE).$(PACKAGE)] section from the DSC file to the output file.
PACKAGE_FILE	L	If defined, then the utility will not generate a package file. The build can then use the value $(PACKAGE_FILE) to have GenFfsFile use an existing package file for creating the firmware file.

Variable Name	G/L	Description
PLATFORM	L	This symbol can be used to provide more selectivity of files in the Platform (DSC) files. If assigned, then the utility will also process any files in the INF file under sections [sources.$(PROCESSOR).$(PLATFORM)], [includes.$(PROCESSOR).$(PLATFORM)], and [libraries.$(PROCESSOR).$(PLATFORM)].
PROCESSOR	G/L	Defines the target processor for which the code is to be built. This symbol will typically be used to include or exclude source files in Platform (DSC) files, and to define the tool chain for building.
SOURCE_DIR	L	For a component, defines the directory of the component source files.

Appendix B
tools_def.txt

The following is the default version of the tools_def.txt file. Note that the lines wrap in this document, however no line wrapping is permitted in the tools_def.txt file.

```
#
#  Copyright (c) 2006-2009, Intel Corporation. All rights reserved.
#  Portions copyright (c) 2008-2009 Apple Inc. All rights reserved.
#
#  This program and the accompanying materials are licensed and made available
#  under the terms and conditions of the BSD License which accompanies this
#  distribution.  The full text of the license may be found at:
#  http://opensource.org/licenses/bsd-license.php
#
#  THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
#  WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.
#

IDENTIFIER = Default TOOL_CHAIN_CONF

# common path macros
DEFINE VS2003_BIN       = C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin
DEFINE VS2003_DLL       = C:\Program Files\Microsoft Visual Studio .NET
2003\Common7\IDE

DEFINE VS2005_BIN       = C:\Program Files\Microsoft Visual Studio 8\Vc\bin
DEFINE VS2005_DLL       = C:\Program Files\Microsoft Visual Studio 8\Common7\IDE
DEFINE VS2005_BINX64    = C:\Program Files\Microsoft Visual Studio
8\Vc\bin\x86_amd64
DEFINE VS2005_BIN64     = C:\Program Files\Microsoft Visual Studio 8\Vc\bin\x86_ia64

DEFINE VS2005x86_BIN    = C:\Program Files (x86)\Microsoft Visual Studio 8\Vc\bin
DEFINE VS2005x86_DLL    = C:\Program Files (x86)\Microsoft Visual Studio
8\Common7\IDE
DEFINE VS2005x86_BINX64 = DEF(VS2005x86_BIN)\x86_amd64
DEFINE VS2005x86_BIN64  = DEF(VS2005x86_BIN)\x86_ia64


# These defines are needed for certain Microsoft Visual Studio tools that
# are used by other toolchains.  An example is that ICC on Windows normally
# uses Microsoft's nmake.exe.

# Some MS_VS_BIN options: DEF(VS2003_BIN), DEF(VS2005_BIN), DEF(VS2005x86_BIN)
DEFINE MS_VS_BIN   = DEF(VS2005_BIN)
# Some MS_VS_DLL options: DEF(VS2003_DLL), DEF(VS2005_DLL), DEF(VS2005x86_DLL)
DEFINE MS_VS_DLL   = DEF(VS2005_DLL)

DEFINE WINDDK_BIN16     = C:\WINDDK\3790.1830\bin\bin16
DEFINE WINDDK_BIN32     = C:\WINDDK\3790.1830\bin\x86
DEFINE WINDDK_BINX64    = C:\WINDDK\3790.1830\bin\win64\x86\amd64
DEFINE WINDDK_BIN64     = C:\WINDDK\3790.1830\bin\win64\x86

# NOTE: The Intel C++ Compiler for Windows requires one of the Microsoft C compiler
#        tool chains for the linker and nmake commands.
#        This configuration assumes a Windows 2003 Server DDK installation.
DEFINE ICC_VERSION      = 9.1
#DEFINE ICC_VERSION     = 10.1.021
DEFINE ICC_BIN32        = C:\Program
Files\Intel\Compiler\C++\DEF(ICC_VERSION)\IA32\Bin
DEFINE ICC_ASM32        = C:\Program
```

```
Files\Intel\Compiler\C++\DEF(ICC_VERSION)\IA32\Bin
DEFINE ICC_BIN32x86     = C:\Program Files
(x86)\Intel\Compiler\C++\DEF(ICC_VERSION)\IA32\Bin
DEFINE ICC_ASM32x86     = C:\Program Files
(x86)\Intel\Compiler\C++\DEF(ICC_VERSION)\IA32\Bin

DEFINE ICC_BINX64       = C:\Program
Files\Intel\Compiler\C++\DEF(ICC_VERSION)\EM64T\Bin
DEFINE ICC_ASMX64       = C:\Program
Files\Intel\Compiler\C++\DEF(ICC_VERSION)\EM64T\Bin
DEFINE ICC_BINX64x86    = C:\Program Files
(x86)\Intel\Compiler\C++\DEF(ICC_VERSION)\EM64T\Bin
DEFINE ICC_ASMX64x86    = C:\Program Files
(x86)\Intel\Compiler\C++\DEF(ICC_VERSION)\EM64T\Bin

DEFINE ICC_BIN64        = C:\Program
Files\Intel\Compiler\C++\DEF(ICC_VERSION)\Itanium\Bin
DEFINE ICC_BIN64x86     = C:\Program Files
(x86)\Intel\Compiler\C++\DEF(ICC_VERSION)\Itanium\Bin

DEFINE EBC_BIN          = C:\Program Files\Intel\EBC\Bin
DEFINE EBC_BINx86       = C:\Program Files (x86)\Intel\EBC\Bin

DEFINE ELFGCC_BIN       = /usr/bin

#
# Option 1: Hard coded full path to compiler suite
DEFINE UNIXGCC_IA32_PETOOLS_PREFIX = /opt/tiano/i386-tiano-pe/i386-tiano-pe/bin/
DEFINE UNIXGCC_X64_PETOOLS_PREFIX  = /opt/tiano/x86_64-pc-mingw64/x86_64-pc-
mingw64/bin/
DEFINE UNIXGCC_IPF_PETOOLS_PREFIX  = /opt/tiano/ia64-pc-elf/ia64-pc-elf/bin/
#
# Option 2: Use an environment variable
#DEFINE UNIXGCC_IA32_PETOOLS_PREFIX = ENV(IA32_PETOOLS_PREFIX)
#DEFINE UNIXGCC_X64_PETOOLS_PREFIX  = ENV(X64_PETOOLS_PREFIX)
#
# Option 3: Install the compiler suite into your default paths
#DEFINE UNIXGCC_IA32_PETOOLS_PREFIX = i386-pc-mingw32-
#DEFINE UNIXGCC_X64_PETOOLS_PREFIX  = x86_64-pc-mingw32-
#
# Option 4: Create links under the BaseTools/Bin/gcc/ARCH directory
# Links needed: gcc, ar & ld
#DEFINE UNIXGCC_IA32_PETOOLS_PREFIX = ENV(WORKSPACE)/BaseTools/Bin/gcc/Ia32/
#DEFINE UNIXGCC_X64_PETOOLS_PREFIX  = ENV(WORKSPACE)/BaseTools/Bin/gcc/X64/
#
# Option 5: Install programs under user's home directory
#DEFINE UNIXGCC_IA32_PETOOLS_PREFIX = ENV(HOME)/programs/gcc/ia32/bin/i686-pc-
mingw32-
#DEFINE UNIXGCC_X64_PETOOLS_PREFIX  = ENV(HOME)/programs/gcc/x64/bin/x86_64-pc-
mingw32-
#

DEFINE CYGWIN_BIN             = c:/cygwin/bin
DEFINE CYGWIN_BINIA32         = c:/cygwin/opt/tiano/i386-tiano-pe/i386-tiano-pe/
bin/
```

```
DEFINE CYGWIN_BINX64           = c:/cygwin/opt/tiano/x86_64-pc-mingw64/x86_64-pc-
mingw64/bin/
DEFINE CYGWIN_BINIPF           = c:/cygwin/opt/tiano/gcc/ipf/bin/ia64-pc-elf-

DEFINE UNIX_IASL_BIN           = /usr/bin/iasl
#DEFINE UNIX_IASL_BIN           = $(HOME)/programs/iasl
DEFINE WIN_ASL_BIN_DIR         = C:\ASL
DEFINE WIN_IASL_BIN            = DEF(WIN_ASL_BIN_DIR)\iasl.exe
DEFINE WIN_ASL_BIN             = DEF(WIN_ASL_BIN_DIR)\asl.exe

DEFINE IASL_OUTFLAGS           = -p
DEFINE MS_ASL_OUTFLAGS         = /Fo=

DEFINE MSFT_ASLPP_FLAGS        = /nologo /EP /C
DEFINE MSFT_ASLCC_FLAGS        = /GL- /Y- /TC /Dmain=ReferenceAcpiTable
DEFINE MSFT_ASLDLINK_FLAGS     = /NODEFAULTLIB /ENTRY:ReferenceAcpiTable /
SUBSYSTEM:CONSOLE

DEFINE ICC_WIN_ASLPP_FLAGS     = /nologo /EP /C
DEFINE ICC_WIN_ASLCC_FLAGS     = /Y- /TC /WX- /Od /Dmain=ReferenceAcpiTable
DEFINE ICC_WIN_ASLDLINK_FLAGS  = /NODEFAULTLIB /ENTRY:ReferenceAcpiTable /
SUBSYSTEM:CONSOLE /NODEFAULTLIB:libmmt /NODEFAULTLIB:libirc

DEFINE IPHONE_TOOLS            = /Developer/Platforms/iPhoneOS.platform/Developer
DEFINE RVCT31_TOOLS_PATH       = c:/Program Files/ARM/RVCT/Programs/3.1/761/win_32-
pentium
DEFINE RVCT31CYGWIN_TOOLS_PATH = /cygdrive/c/Program\ Files/ARM/RVCT/Programs/3.1/
761/win_32-pentium

# Update to the location of the gcc executables
DEFINE ARMGCC_BIN              = /

################################################################################
#
#
# format: TARGET_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE = <string>
# priority:
#         TARGET_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE (Highest)
#         ******_TOOLCHAIN_ARCH_COMMANDTYPE_ATTRIBUTE
#         TARGET_*********_ARCH_COMMANDTYPE_ATTRIBUTE
#         ******_*********_ARCH_COMMANDTYPE_ATTRIBUTE
#         TARGET_TOOLCHAIN_****_COMMANDTYPE_ATTRIBUTE
#         ******_TOOLCHAIN_****_COMMANDTYPE_ATTRIBUTE
#         TARGET_*********_****_COMMANDTYPE_ATTRIBUTE
#         ******_*********_****_COMMANDTYPE_ATTRIBUTE
#         TARGET_TOOLCHAIN_ARCH_***********_ATTRIBUTE
#         ******_TOOLCHAIN_ARCH_***********_ATTRIBUTE
#         TARGET_*********_ARCH_***********_ATTRIBUTE
#         ******_*********_ARCH_***********_ATTRIBUTE
#         TARGET_TOOLCHAIN_****_***********_ATTRIBUTE
#         ******_TOOLCHAIN_****_***********_ATTRIBUTE
#         TARGET_*********_****_***********_ATTRIBUTE
#         ******_*********_****_***********_ATTRIBUTE (Lowest)
#
################################################################################
```

```
#
################################################################################
#
#
# Supported Tool Chains
# =====================
#   VS2003                  - win32 - Microsoft Visual Studio .NET 2003, Intel EBC,
Intel ASL
#   VS2005                - win32 - Microsoft Visual Studio 2005 Team Suite Edition,
Intel EBC, Intel ASL (also compatible with VS 2005 Express, Standard, and Pro)
#   DDK3790               - win32 - Microsoft Windows DDK 3790.1830, Intel EBC, Intel
ASL
#   UNIXGCC                 - UNIX GCC, No EBC, Intel ASL
#   ELFGCC                  - Linux ELF GCC, No EBC, Intel ASL
#   CYGGCC                  - win32 - CygWin GCC, Intel EBC, Intel ASL
#   ICC                     - win32 - Intel C Compiler V9.1, Intel EBC, Intel ASL
#   MYTOOLS                 - win32 - VS2005 for IA32/X64, WINDDK 3790.1830 for IPF,
Intel EBC, Intel ASL
#   VS2003xASL              - win32 - Microsoft Visual Studio .NET 2003, Intel EBC,
Microsoft ASL
#   VS2005xASL            - win32 - Microsoft Visual Studio 2005 Team Suite Edition,
Intel EBC, Microsoft ASL
#   DDK3790xASL             - win32 - Microsoft Windows DDK 3790.1830, Intel EBC,
Microsoft ASL
#   CYGGCCxASL              - win32 - CygWin GCC, Intel EBC, Microsoft ASL
#   ICCxASL                 - win32 - Intel C Compiler V9.1, Intel EBC, Microsoft ASL
#   VS2005x86             - win64 - Microsoft Visual Studio 2005 Team Suite Edition
(x86), Intel EBC, Intel ASL (also compatible with VS 2005 Express, Standard, and
Pro)
#   ICCx86                - win64 - Intel C Compiler V9.1 (x86), Intel EBC, Intel ASL
#   VS2005x86xASL          - win64 - Microsoft Visual Studio 2005 Team Suite Edition
(x86), Intel EBC, Microsoft ASL (also compatible with VS 2005 Express, Standard, and
Pro)
#   ICCx86xASL            - win64 - Intel C Compiler V9.1 (x86), Intel EBC, Microsoft
ASL
#   CYGGCCx86               - win64 - CygWin GCC (x86), Intel EBC (x86), Intel ASL
#   CYGGCCx86xASL          - win64 - CygWin GCC (x86), Intel EBC (x86), Microsoft ASL
# * Commented out - All versions of VS2005 use the same standard install directory
#
################################################################################
#
################################################################################
#
#
# Supported Tool Chain Family
# ===========================
#   MSFT            - Microsoft
#   GCC             - GNU GCC
#   INTEL           - INTEL
################################################################################
#
################################################################################
#
#
# Intel EFI Byte Code Compiler (Template)
```

```
#
################################################################################
#
# *_*_EBC_*_FAMILY                    = INTEL
#
# *_*_EBC_PP_PATH                     = C:\Program Files\Intel\EBC\Bin\iec.exe
# *_*_EBC_CC_PATH                     = C:\Program Files\Intel\EBC\Bin\iec.exe
# *_*_EBC_SLINK_PATH                  = C:\Program Files\Intel\EBC\Bin\link.exe
#
# *_*_EBC_SLINK_FLAGS                 = /lib /NOLOGO /MACHINE:EBC
# *_*_EBC_PP_FLAGS                   = /nologo /E /TC /FI$(DEST_DIR_DEBUG)/AutoGen.h
# *_*_EBC_CC_FLAGS                  = /nologo /FAcs /c /W3 /WX /FI$(DEST_DIR_DEBUG)/
AutoGen.h
# *_*_EBC_DLINK_FLAGS                 = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /MACHINE:EBC /OPT:REF /NODEFAULTLIB /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER
#
################################################################################
#
#
# Intel ACPI Source Language Compiler (Template)
#
################################################################################
#
# *_*_*_ASL_FAMILY                    = INTEL
#
# *_*_*_ASL_PATH                      = C:\ASL\iasl.exe
#
################################################################################
#
#
# Microsoft ACPI Source Language Compiler (Template)
#
################################################################################
#
# *_*_*_ASL_FAMILY                    = MSFT
#
# *_*_*_ASL_PATH                      = C:\ASL\asl.exe
#
################################################################################
#


################################################################################
#
#
# Microsoft Visual Studio .NET 2003 (IA-32 only, with Link Time Code Generation)
# And Intel ACPI Compiler
#
################################################################################
#
#   VS2003           - Microsoft Visual Studio .NET 2003 and Intel ACPI Source
Language Compiler (iasl.exe)
*_VS2003_*_*_FAMILY                 = MSFT
```

```
##################
# ASL definitions
##################
*_VS2003_*_ASL_PATH                = DEF(WIN_IASL_BIN)
*_VS2003_*_ASL_FLAGS               =
*_VS2003_*_ASL_OUTFLAGS            = DEF(IASL_OUTFLAGS)
*_VS2003_*_ASLCC_FLAGS             = DEF(MSFT_ASLCC_FLAGS)
*_VS2003_*_ASLPP_FLAGS             = DEF(MSFT_ASLPP_FLAGS)
*_VS2003_*_ASLDLINK_FLAGS          = DEF(MSFT_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_VS2003_IA32_*_DLL                = DEF(VS2003_DLL)

*_VS2003_IA32_MAKE_PATH            = DEF(VS2003_BIN)\nmake.exe
*_VS2003_IA32_CC_PATH              = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_VFRPP_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_SLINK_PATH           = DEF(VS2003_BIN)\lib.exe
*_VS2003_IA32_DLINK_PATH           = DEF(VS2003_BIN)\link.exe
*_VS2003_IA32_ASMLINK_PATH         = DEF(WINDDK_BIN16)\link16.exe
*_VS2003_IA32_APP_PATH             = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_PP_PATH              = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_ASM_PATH             = DEF(VS2003_BIN)\ml.exe
*_VS2003_IA32_ASLCC_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_ASLPP_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003_IA32_ASLDLINK_PATH        = DEF(VS2003_BIN)\link.exe
*_VS2003_IA32_RC_PATH              = DEF(VS2003_BIN)\rc.exe

      *_VS2003_IA32_MAKE_FLAGS     = /nologo
      *_VS2003_IA32_APP_FLAGS      = /nologo /E /TC
      *_VS2003_IA32_PP_FLAGS       = /nologo /E /TC /FIAutoGen.h
      *_VS2003_IA32_VFRPP_FLAGS    = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
  DEBUG_VS2003_IA32_CC_FLAGS       = /nologo /c /WX /W4 /Gs8192 /Gy /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /GX- /Zi /Gm
RELEASE_VS2003_IA32_CC_FLAGS       = /nologo /c /WX /W4 /Gs8192 /Gy /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /GX-
  DEBUG_VS2003_IA32_ASM_FLAGS      = /nologo /c /WX /W3 /coff /Cx /Zd /Zi
RELEASE_VS2003_IA32_ASM_FLAGS      = /nologo /c /WX /W3 /coff /Cx /Zd
      *_VS2003_IA32_SLINK_FLAGS    = /nologo /LTCG
  DEBUG_VS2003_IA32_DLINK_FLAGS    = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2003_IA32_DLINK_FLAGS    = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2003_IA32_ASMLINK_FLAGS   = /nologo /tiny

##################
# EBC definitions
##################
*_VS2003_EBC_*_FAMILY               = INTEL

*_VS2003_EBC_MAKE_PATH              = DEF(VS2003_BIN)\nmake.exe
*_VS2003_EBC_PP_PATH                = DEF(EBC_BIN)\iec.exe
*_VS2003_EBC_VFRPP_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2003_EBC_CC_PATH                = DEF(EBC_BIN)\iec.exe
*_VS2003_EBC_SLINK_PATH             = DEF(EBC_BIN)\link.exe
*_VS2003_EBC_DLINK_PATH             = DEF(EBC_BIN)\link.exe
*_VS2003_EBC_RC_PATH                = DEF(VS2003_BIN)\rc.exe

*_VS2003_EBC_MAKE_FLAGS             = /nologo
*_VS2003_EBC_PP_FLAGS               = /nologo /E /TC /FIAutoGen.h
*_VS2003_EBC_CC_FLAGS               = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2003_EBC_VFRPP_FLAGS            = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2003_EBC_SLINK_FLAGS            = /lib /NOLOGO /MACHINE:EBC
*_VS2003_EBC_DLINK_FLAGS            = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

################################################################################
#
#
#   Microsoft Visual Studio .NET 2003 and Microsoft ACPI compiler
#
################################################################################
#
#   VS2003xASL          - Microsoft Visual Studio .NET 2003 and Microsoft ACPI
Source Language Compiler (asl.exe)
*_VS2003xASL_*_*_FAMILY                 = MSFT

##################
# ASL definitions
##################
*_VS2003xASL_*_ASL_PATH                 = DEF(WIN_ASL_BIN)
*_VS2003xASL_*_ASL_FLAGS                =
*_VS2003xASL_*_ASL_OUTFLAGS             = DEF(MS_ASL_OUTFLAGS)
*_VS2003xASL_*_ASLCC_FLAGS              = DEF(MSFT_ASLCC_FLAGS)
*_VS2003xASL_*_ASLPP_FLAGS              = DEF(MSFT_ASLPP_FLAGS)
*_VS2003xASL_*_ASLDLINK_FLAGS           = DEF(MSFT_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_VS2003xASL_IA32_*_DLL                 = DEF(VS2003_DLL)

*_VS2003xASL_IA32_MAKE_PATH             = DEF(VS2003_BIN)\nmake.exe
```

```
*_VS2003xASL_IA32_CC_PATH              = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_VFRPP_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_SLINK_PATH           = DEF(VS2003_BIN)\lib.exe
*_VS2003xASL_IA32_DLINK_PATH           = DEF(VS2003_BIN)\link.exe
*_VS2003xASL_IA32_ASMLINK_PATH         = DEF(WINDDK_BIN16)\link16.exe
*_VS2003xASL_IA32_APP_PATH             = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_PP_PATH              = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_ASM_PATH             = DEF(VS2003_BIN)\ml.exe
*_VS2003xASL_IA32_ASLCC_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_ASLPP_PATH           = DEF(VS2003_BIN)\cl.exe
*_VS2003xASL_IA32_ASLDLINK_PATH        = DEF(VS2003_BIN)\link.exe
*_VS2003xASL_IA32_RC_PATH              = DEF(VS2003_BIN)\rc.exe


      *_VS2003xASL_IA32_MAKE_FLAGS     = /nologo
      *_VS2003xASL_IA32_APP_FLAGS      = /nologo /E /TC
      *_VS2003xASL_IA32_PP_FLAGS       = /nologo /E /TC /FIAutoGen.h
      *_VS2003xASL_IA32_VFRPP_FLAGS    = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
  DEBUG_VS2003xASL_IA32_CC_FLAGS       = /nologo /c /WX /W4 /Gs8192 /Gy /D UNICODE
/O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /GX- /Zi /Gm
RELEASE_VS2003xASL_IA32_CC_FLAGS       = /nologo /c /WX /W4 /Gs8192 /Gy /D UNICODE
/O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /GX-
  DEBUG_VS2003xASL_IA32_ASM_FLAGS      = /nologo /c /WX /W3 /coff /Cx /Zd /Zi
RELEASE_VS2003xASL_IA32_ASM_FLAGS      = /nologo /c /WX /W3 /coff /Cx /Zd
      *_VS2003xASL_IA32_SLINK_FLAGS    = /nologo /LTCG
  DEBUG_VS2003xASL_IA32_DLINK_FLAGS    = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /DEBUG
RELEASE_VS2003xASL_IA32_DLINK_FLAGS    = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
```

```
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2003xASL_IA32_ASMLINK_FLAGS   = /nologo /tiny

###################
# EBC definitions
###################
*_VS2003xASL_EBC_*_FAMILY               = INTEL

*_VS2003xASL_EBC_MAKE_PATH              = DEF(VS2003_BIN)\nmake.exe
*_VS2003xASL_EBC_PP_PATH                = DEF(EBC_BIN)\iec.exe
*_VS2003xASL_EBC_VFRPP_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2003xASL_EBC_CC_PATH                = DEF(EBC_BIN)\iec.exe
*_VS2003xASL_EBC_SLINK_PATH             = DEF(EBC_BIN)\link.exe
*_VS2003xASL_EBC_DLINK_PATH             = DEF(EBC_BIN)\link.exe
*_VS2003xASL_EBC_RC_PATH                = DEF(VS2003_BIN)\rc.exe

*_VS2003xASL_EBC_MAKE_FLAGS             = /nologo
*_VS2003xASL_EBC_PP_FLAGS               = /nologo /E /TC /FIAutoGen.h
*_VS2003xASL_EBC_CC_FLAGS               = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2003xASL_EBC_VFRPP_FLAGS            = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2003xASL_EBC_SLINK_FLAGS            = /lib /NOLOGO /MACHINE:EBC
*_VS2003xASL_EBC_DLINK_FLAGS            = "C:\Program
Files\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

##############################################################################
#
#
# Microsoft Visual Studio 2005
#
#   VS2005  - Microsoft Visual Studio 2005 All Edition, including Standard,
Professional, Express, TeamSuite
#   ASL     - Intel ACPI Source Language Compiler
##############################################################################
#
#   VS2005          - Microsoft Visual Studio 2005 ALL Edition, including Standard,
Professional, Express, TeamSuite
*_VS2005_*_*_FAMILY               = MSFT

*_VS2005_*_MAKE_PATH              = DEF(VS2005_BIN)\nmake.exe
*_VS2005_*_MAKE_FLAGS             = /nologo
*_VS2005_*_RC_PATH                = DEF(VS2005_BIN)\rc.exe

*_VS2005_*_SLINK_FLAGS            = /NOLOGO /LTCG
*_VS2005_*_APP_FLAGS              = /nologo /E /TC
*_VS2005_*_PP_FLAGS               = /nologo /E /TC /FIAutoGen.h
*_VS2005_*_VFRPP_FLAGS            = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
```

```
*_VS2005_*_ASL_PATH               = DEF(WIN_IASL_BIN)
*_VS2005_*_ASL_FLAGS              =
*_VS2005_*_ASL_OUTFLAGS           = DEF(IASL_OUTFLAGS)
*_VS2005_*_ASLCC_FLAGS            = DEF(MSFT_ASLCC_FLAGS)
*_VS2005_*_ASLPP_FLAGS            = DEF(MSFT_ASLPP_FLAGS)
*_VS2005_*_ASLDLINK_FLAGS         = DEF(MSFT_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_VS2005_IA32_*_DLL               = DEF(VS2005_DLL)

*_VS2005_IA32_MAKE_PATH           = DEF(VS2005_BIN)\nmake.exe
*_VS2005_IA32_CC_PATH             = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_VFRPP_PATH          = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_SLINK_PATH          = DEF(VS2005_BIN)\lib.exe
*_VS2005_IA32_DLINK_PATH          = DEF(VS2005_BIN)\link.exe
*_VS2005_IA32_ASMLINK_PATH        = DEF(WINDDK_BIN16)\link16.exe
*_VS2005_IA32_APP_PATH            = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_PP_PATH             = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_ASM_PATH            = DEF(VS2005_BIN)\ml.exe
*_VS2005_IA32_ASLCC_PATH          = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_ASLPP_PATH          = DEF(VS2005_BIN)\cl.exe
*_VS2005_IA32_ASLDLINK_PATH       = DEF(VS2005_BIN)\link.exe

      *_VS2005_IA32_MAKE_FLAGS    = /nologo
  DEBUG_VS2005_IA32_CC_FLAGS     = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /O1ib2
/GL /FIAutoGen.h /EHs-c- /GR- /GF /Gy /Zi /Gm
RELEASE_VS2005_IA32_CC_FLAGS      = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005_IA32_ASM_FLAGS     = /nologo /c /WX /W3 /Cx /coff /Zd /Zi
RELEASE_VS2005_IA32_ASM_FLAGS     = /nologo /c /WX /W3 /Cx /coff /Zd
  DEBUG_VS2005_IA32_DLINK_FLAGS   = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005_IA32_DLINK_FLAGS   = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2005_IA32_ASMLINK_FLAGS = /nologo /tiny

###################
# X64 definitions
###################
*_VS2005_X64_*_DLL         = DEF(VS2005_DLL)

*_VS2005_X64_CC_PATH       = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_PP_PATH       = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_APP_PATH      = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_VFRPP_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_ASM_PATH      = DEF(VS2005_BINX64)\ml64.exe
*_VS2005_X64_SLINK_PATH    = DEF(VS2005_BINX64)\lib.exe
*_VS2005_X64_DLINK_PATH    = DEF(VS2005_BINX64)\link.exe
*_VS2005_X64_ASLCC_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_ASLPP_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005_X64_ASLDLINK_PATH = DEF(VS2005_BINX64)\link.exe


  DEBUG_VS2005_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_VS2005_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_VS2005_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_VS2005_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_VS2005_IPF_*_DLL         = DEF(VS2005_DLL)

*_VS2005_IPF_PP_PATH       = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_APP_PATH      = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_VFRPP_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_CC_PATH       = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_ASM_PATH      = DEF(VS2005_BIN64)\ias.exe
*_VS2005_IPF_SLINK_PATH    = DEF(VS2005_BIN64)\lib.exe
*_VS2005_IPF_DLINK_PATH    = DEF(VS2005_BIN64)\link.exe
*_VS2005_IPF_ASLCC_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_ASLPP_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005_IPF_ASLDLINK_PATH = DEF(VS2005_BIN64)\link.exe

  DEBUG_VS2005_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os /
GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_VS2005_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os /
GL /FIAutoGen.h /QIPF_fr32
  DEBUG_VS2005_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d debug
RELEASE_VS2005_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_VS2005_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF,ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb /DEBUG
RELEASE_VS2005_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF.ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
```

```
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb

##################
# EBC definitions
##################
*_VS2005_EBC_*_FAMILY            = INTEL

*_VS2005_EBC_MAKE_PATH           = DEF(VS2005_BIN)\nmake.exe
*_VS2005_EBC_PP_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2005_EBC_VFRPP_PATH          = DEF(EBC_BIN)\iec.exe
*_VS2005_EBC_CC_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2005_EBC_SLINK_PATH          = DEF(EBC_BIN)\link.exe
*_VS2005_EBC_DLINK_PATH          = DEF(EBC_BIN)\link.exe

*_VS2005_EBC_MAKE_FLAGS          = /nologo
*_VS2005_EBC_PP_FLAGS            = /nologo /E /TC /FIAutoGen.h
*_VS2005_EBC_CC_FLAGS            = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2005_EBC_VFRPP_FLAGS         = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2005_EBC_SLINK_FLAGS         = /lib /NOLOGO /MACHINE:EBC
*_VS2005_EBC_DLINK_FLAGS         = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

###############################################################################
#
#
# Microsoft Visual Studio 2005
#
#   VS2005  - Microsoft Visual Studio 2005 All Edition, including Standard,
Professional, Express, TeamSuite
#   ASL     - Microsoft ACPI Source Language Compiler (asl.exe)
###############################################################################
#
#   VS2005xASL          - Microsoft Visual Studio 2005 ALL Edition, including
Standard, Professional, Express, TeamSuite
*_VS2005xASL_*_*_FAMILY          = MSFT

*_VS2005xASL_*_MAKE_PATH         = DEF(VS2005_BIN)\nmake.exe
*_VS2005xASL_*_MAKE_FLAG         = /nologo
*_VS2005xASL_*_RC_PATH           = DEF(VS2005_BIN)\rc.exe

*_VS2005xASL_*_SLINK_FLAGS       = /NOLOGO /LTCG
*_VS2005xASL_*_APP_FLAGS         = /nologo /E /TC
*_VS2005xASL_*_PP_FLAGS          = /nologo /E /TC /FIAutoGen.h
*_VS2005xASL_*_VFRPP_FLAGS       = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

##################
# ASL definitions
##################
*_VS2005xASL_*_ASL_PATH          = DEF(WIN_ASL_BIN)
*_VS2005xASL_*_ASL_FLAGS         =
```

```
*_VS2005xASL_*_ASL_OUTFLAGS    = DEF(MS_ASL_OUTFLAGS)
*_VS2005xASL_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_VS2005xASL_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_VS2005xASL_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_VS2005xASL_IA32_*_DLL        = DEF(VS2005_DLL)

*_VS2005xASL_IA32_MAKE_PATH    = DEF(VS2005_BIN)\nmake.exe
*_VS2005xASL_IA32_CC_PATH      = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_VFRPP_PATH   = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_SLINK_PATH   = DEF(VS2005_BIN)\lib.exe
*_VS2005xASL_IA32_DLINK_PATH   = DEF(VS2005_BIN)\link.exe
*_VS2005xASL_IA32_ASMLINK_PATH = DEF(WINDDK_BIN16)\link16.exe
*_VS2005xASL_IA32_APP_PATH     = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_PP_PATH      = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_ASM_PATH     = DEF(VS2005_BIN)\ml.exe
*_VS2005xASL_IA32_ASLCC_PATH    = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_ASLPP_PATH    = DEF(VS2005_BIN)\cl.exe
*_VS2005xASL_IA32_ASLDLINK_PATH = DEF(VS2005_BIN)\link.exe

      *_VS2005xASL_IA32_MAKE_FLAGS  = /nologo
  DEBUG_VS2005xASL_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Gy /Zi /Gm
RELEASE_VS2005xASL_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd /Zi
RELEASE_VS2005xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd
  DEBUG_VS2005xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2005xASL_IA32_ASMLINK_FLAGS= /nologo /tiny

###################
# X64 definitions
###################
*_VS2005xASL_X64_*_DLL         = DEF(VS2005_DLL)

*_VS2005xASL_X64_CC_PATH       = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_PP_PATH       = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_APP_PATH      = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_VFRPP_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_ASM_PATH      = DEF(VS2005_BINX64)\ml64.exe
*_VS2005xASL_X64_SLINK_PATH    = DEF(VS2005_BINX64)\lib.exe
*_VS2005xASL_X64_DLINK_PATH    = DEF(VS2005_BINX64)\link.exe
*_VS2005xASL_X64_ASLCC_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_ASLPP_PATH    = DEF(VS2005_BINX64)\cl.exe
*_VS2005xASL_X64_ASLDLINK_PATH = DEF(VS2005_BINX64)\link.exe


  DEBUG_VS2005xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE
/O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_VS2005xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE
/O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_VS2005xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_VS2005xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_VS2005xASL_IPF_*_DLL         = DEF(VS2005_DLL)

*_VS2005xASL_IPF_PP_PATH       = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_APP_PATH      = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_VFRPP_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_CC_PATH       = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_ASM_PATH      = DEF(VS2005_BIN64)\ias.exe
*_VS2005xASL_IPF_SLINK_PATH    = DEF(VS2005_BIN64)\lib.exe
*_VS2005xASL_IPF_DLINK_PATH    = DEF(VS2005_BIN64)\link.exe
*_VS2005xASL_IPF_ASLCC_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_ASLPP_PATH    = DEF(VS2005_BIN64)\cl.exe
*_VS2005xASL_IPF_ASLDLINK_PATH = DEF(VS2005_BIN64)\link.exe

  DEBUG_VS2005xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /
Os /GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_VS2005xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /
Os /GL /FIAutoGen.h /QIPF_fr32
  DEBUG_VS2005xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d debug
RELEASE_VS2005xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_VS2005xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF,ICF
/IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb /DEBUG
RELEASE_VS2005xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF.ICF
/IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
```

```
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb

##################
# EBC definitions
##################
*_VS2005xASL_EBC_*_FAMILY            = INTEL

*_VS2005xASL_EBC_MAKE_PATH           = DEF(VS2005_BIN)\nmake.exe
*_VS2005xASL_EBC_PP_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2005xASL_EBC_VFRPP_PATH          = DEF(EBC_BIN)\iec.exe
*_VS2005xASL_EBC_CC_PATH             = DEF(EBC_BIN)\iec.exe
*_VS2005xASL_EBC_SLINK_PATH          = DEF(EBC_BIN)\link.exe
*_VS2005xASL_EBC_DLINK_PATH          = DEF(EBC_BIN)\link.exe

*_VS2005xASL_EBC_MAKE_FLAGS          = /nologo
*_VS2005xASL_EBC_PP_FLAGS            = /nologo /E /TC /FIAutoGen.h
*_VS2005xASL_EBC_CC_FLAGS            = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2005xASL_EBC_VFRPP_FLAGS         = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2005xASL_EBC_SLINK_FLAGS         = /lib /NOLOGO /MACHINE:EBC
*_VS2005xASL_EBC_DLINK_FLAGS        = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

################################################################################
#
#
# Microsoft Visual Studio 2005 (x86)
#
#   VS2005  - Microsoft Visual Studio 2005 All Edition, including Standard,
Professional, Express, TeamSuite
#   ASL     - Intel ACPI Source Language Compiler
################################################################################
#
#   VS2005x86           - Microsoft Visual Studio 2005 ALL Edition, including
Standard, Professional, Express, TeamSuite
*_VS2005x86_*_*_FAMILY        = MSFT

*_VS2005x86_*_MAKE_PATH       = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86_*_MAKE_FLAGS      = /nologo
*_VS2005x86_*_RC_PATH         = DEF(VS2005x86_BIN)\rc.exe

*_VS2005x86_*_SLINK_FLAGS     = /NOLOGO /LTCG
*_VS2005x86_*_APP_FLAGS       = /nologo /E /TC
*_VS2005x86_*_PP_FLAGS        = /nologo /E /TC /FIAutoGen.h
*_VS2005x86_*_VFRPP_FLAGS     = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

##################
# ASL definitions
##################
*_VS2005x86_*_ASL_PATH        = DEF(WIN_IASL_BIN)
*_VS2005x86_*_ASL_FLAGS       =
```

```
*_VS2005x86_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
*_VS2005x86_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_VS2005x86_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_VS2005x86_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_VS2005x86_IA32_*_DLL        = DEF(VS2005x86_DLL)

*_VS2005x86_IA32_MAKE_PATH    = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86_IA32_CC_PATH      = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_VFRPP_PATH   = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_SLINK_PATH   = DEF(VS2005x86_BIN)\lib.exe
*_VS2005x86_IA32_DLINK_PATH   = DEF(VS2005x86_BIN)\link.exe
*_VS2005x86_IA32_ASMLINK_PATH = DEF(WINDDK_BIN16)\link16.exe
*_VS2005x86_IA32_APP_PATH     = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_PP_PATH      = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_ASM_PATH     = DEF(VS2005x86_BIN)\ml.exe
*_VS2005x86_IA32_ASLCC_PATH    = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_ASLPP_PATH    = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86_IA32_ASLDLINK_PATH = DEF(VS2005x86_BIN)\link.exe

      *_VS2005x86_IA32_MAKE_FLAGS  = /nologo
  DEBUG_VS2005x86_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Gy /Zi /Gm
RELEASE_VS2005x86_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005x86_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd /Zi
RELEASE_VS2005x86_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd
  DEBUG_VS2005x86_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005x86_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2005x86_IA32_ASMLINK_FLAGS= /nologo /tiny

###################
# X64 definitions
###################
*_VS2005x86_X64_*_DLL         = DEF(VS2005x86_DLL)

*_VS2005x86_X64_CC_PATH       = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_PP_PATH       = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_APP_PATH      = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_VFRPP_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_ASM_PATH      = DEF(VS2005x86_BINX64)\ml64.exe
*_VS2005x86_X64_SLINK_PATH    = DEF(VS2005x86_BINX64)\lib.exe
*_VS2005x86_X64_DLINK_PATH    = DEF(VS2005x86_BINX64)\link.exe
*_VS2005x86_X64_ASLCC_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_ASLPP_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86_X64_ASLDLINK_PATH = DEF(VS2005x86_BINX64)\link.exe


  DEBUG_VS2005x86_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_VS2005x86_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005x86_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_VS2005x86_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_VS2005x86_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005x86_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_VS2005x86_IPF_*_DLL         = DEF(VS2005x86_DLL)

*_VS2005x86_IPF_PP_PATH       = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_APP_PATH      = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_VFRPP_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_CC_PATH       = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_ASM_PATH      = DEF(VS2005x86_BIN64)\ias.exe
*_VS2005x86_IPF_SLINK_PATH    = DEF(VS2005x86_BIN64)\lib.exe
*_VS2005x86_IPF_DLINK_PATH    = DEF(VS2005x86_BIN64)\link.exe
*_VS2005x86_IPF_ASLCC_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_ASLPP_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86_IPF_ASLDLINK_PATH = DEF(VS2005x86_BIN64)\link.exe

  DEBUG_VS2005x86_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os
/GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_VS2005x86_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os
/GL /FIAutoGen.h /QIPF_fr32
  DEBUG_VS2005x86_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d debug
RELEASE_VS2005x86_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_VS2005x86_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF,ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb /DEBUG
RELEASE_VS2005x86_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF.ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
```

```
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb

###################
# EBC definitions
###################
*_VS2005x86_EBC_*_FAMILY            = INTEL

*_VS2005x86_EBC_MAKE_PATH           = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86_EBC_PP_PATH             = DEF(EBC_BINx86)\iec.exe
*_VS2005x86_EBC_VFRPP_PATH          = DEF(EBC_BINx86)\iec.exe
*_VS2005x86_EBC_CC_PATH             = DEF(EBC_BINx86)\iec.exe
*_VS2005x86_EBC_SLINK_PATH          = DEF(EBC_BINx86)\link.exe
*_VS2005x86_EBC_DLINK_PATH          = DEF(EBC_BINx86)\link.exe

*_VS2005x86_EBC_MAKE_FLAGS          = /nologo
*_VS2005x86_EBC_PP_FLAGS            = /nologo /E /TC /FIAutoGen.h
*_VS2005x86_EBC_CC_FLAGS            = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2005x86_EBC_VFRPP_FLAGS         = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2005x86_EBC_SLINK_FLAGS         = /lib /NOLOGO /MACHINE:EBC
*_VS2005x86_EBC_DLINK_FLAGS         = "C:\Program Files
(x86)\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

##############################################################################
#
#
# Microsoft Visual Studio 2005 (x86)
#
#   VS2005  - Microsoft Visual Studio 2005 All Edition, including Standard,
Professional, Express, TeamSuite
#   ASL     - Microsoft ACPI Source Language Compiler
##############################################################################
#
#   VS2005x86xASL           - Microsoft Visual Studio 2005 ALL Edition, including
Standard, Professional, Express, TeamSuite
*_VS2005x86xASL_*_*_FAMILY        = MSFT

*_VS2005x86xASL_*_MAKE_PATH       = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86xASL_*_MAKE_FLAGS      = /nologo
*_VS2005x86xASL_*_RC_PATH         = DEF(VS2005x86_BIN)\rc.exe

*_VS2005x86xASL_*_SLINK_FLAGS     = /NOLOGO /LTCG
*_VS2005x86xASL_*_APP_FLAGS       = /nologo /E /TC
*_VS2005x86xASL_*_PP_FLAGS        = /nologo /E /TC /FIAutoGen.h
*_VS2005x86xASL_*_VFRPP_FLAGS     = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
*_VS2005x86xASL_*_ASL_PATH        = DEF(WIN_ASL_BIN)
*_VS2005x86xASL_*_ASL_FLAGS       =
```

```
*_VS2005x86xASL_*_ASL_OUTFLAGS    = DEF(MS_ASL_OUTFLAGS)
*_VS2005x86xASL_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_VS2005x86xASL_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_VS2005x86xASL_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_VS2005x86xASL_IA32_*_DLL        = DEF(VS2005x86_DLL)

*_VS2005x86xASL_IA32_MAKE_PATH    = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86xASL_IA32_CC_PATH      = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_VFRPP_PATH   = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_SLINK_PATH   = DEF(VS2005x86_BIN)\lib.exe
*_VS2005x86xASL_IA32_DLINK_PATH   = DEF(VS2005x86_BIN)\link.exe
*_VS2005x86xASL_IA32_ASMLINK_PATH = DEF(WINDDK_BIN16)\link16.exe
*_VS2005x86xASL_IA32_APP_PATH     = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_PP_PATH      = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_ASM_PATH     = DEF(VS2005x86_BIN)\ml.exe
*_VS2005x86xASL_IA32_ASLCC_PATH    = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_ASLPP_PATH    = DEF(VS2005x86_BIN)\cl.exe
*_VS2005x86xASL_IA32_ASLDLINK_PATH = DEF(VS2005x86_BIN)\link.exe

      *_VS2005x86xASL_IA32_MAKE_FLAGS  = /nologo
  DEBUG_VS2005x86xASL_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE
/O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Gy /Zi /Gm
RELEASE_VS2005x86xASL_IA32_CC_FLAGS    = /nologo /c /WX /GS- /W4 /Gs8192 /D UNICODE
/O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005x86xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd /Zi
RELEASE_VS2005x86xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /Cx /coff /Zd
  DEBUG_VS2005x86xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF
/OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005x86xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
```

```
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_VS2005x86xASL_IA32_ASMLINK_FLAGS= /nologo /tiny

###################
# X64 definitions
###################
*_VS2005x86xASL_X64_*_DLL         = DEF(VS2005x86_DLL)

*_VS2005x86xASL_X64_CC_PATH       = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_PP_PATH       = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_APP_PATH      = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_VFRPP_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_ASM_PATH      = DEF(VS2005x86_BINX64)\ml64.exe
*_VS2005x86xASL_X64_SLINK_PATH    = DEF(VS2005x86_BINX64)\lib.exe
*_VS2005x86xASL_X64_DLINK_PATH    = DEF(VS2005x86_BINX64)\link.exe
*_VS2005x86xASL_X64_ASLCC_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_ASLPP_PATH    = DEF(VS2005x86_BINX64)\cl.exe
*_VS2005x86xASL_X64_ASLDLINK_PATH = DEF(VS2005x86_BINX64)\link.exe


  DEBUG_VS2005x86xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_VS2005x86xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_VS2005x86xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_VS2005x86xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_VS2005x86xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF
/OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG
/DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_VS2005x86xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF
/OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG
```

```
/DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_VS2005x86xASL_IPF_*_DLL         = DEF(VS2005x86_DLL)

*_VS2005x86xASL_IPF_PP_PATH       = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_APP_PATH      = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_VFRPP_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_CC_PATH       = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_ASM_PATH      = DEF(VS2005x86_BIN64)\ias.exe
*_VS2005x86xASL_IPF_SLINK_PATH    = DEF(VS2005x86_BIN64)\lib.exe
*_VS2005x86xASL_IPF_DLINK_PATH    = DEF(VS2005x86_BIN64)\link.exe
*_VS2005x86xASL_IPF_ASLCC_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_ASLPP_PATH    = DEF(VS2005x86_BIN64)\cl.exe
*_VS2005x86xASL_IPF_ASLDLINK_PATH = DEF(VS2005x86_BIN64)\link.exe

  DEBUG_VS2005x86xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy
/Os /GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_VS2005x86xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy
/Os /GL /FIAutoGen.h /QIPF_fr32
  DEBUG_VS2005x86xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d
debug
RELEASE_VS2005x86xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_VS2005x86xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/
$(BASE_NAME).pdb /DEBUG
RELEASE_VS2005x86xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF.ICF /IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEBUG_DIR)/$(BASE_NAME).map /PDB:$(DEBUG_DIR)/
$(BASE_NAME).pdb

##################
# EBC definitions
##################
*_VS2005x86xASL_EBC_*_FAMILY            = INTEL

*_VS2005x86xASL_EBC_MAKE_PATH           = DEF(VS2005x86_BIN)\nmake.exe
*_VS2005x86xASL_EBC_PP_PATH             = DEF(EBC_BINx86)\iec.exe
*_VS2005x86xASL_EBC_VFRPP_PATH          = DEF(EBC_BINx86)\iec.exe
*_VS2005x86xASL_EBC_CC_PATH             = DEF(EBC_BINx86)\iec.exe
*_VS2005x86xASL_EBC_SLINK_PATH          = DEF(EBC_BINx86)\link.exe
*_VS2005x86xASL_EBC_DLINK_PATH          = DEF(EBC_BINx86)\link.exe

*_VS2005x86xASL_EBC_MAKE_FLAGS          = /nologo
*_VS2005x86xASL_EBC_PP_FLAGS            = /nologo /E /TC /FIAutoGen.h
*_VS2005x86xASL_EBC_CC_FLAGS            = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_VS2005x86xASL_EBC_VFRPP_FLAGS         = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_VS2005x86xASL_EBC_SLINK_FLAGS         = /lib /NOLOGO /MACHINE:EBC
*_VS2005x86xASL_EBC_DLINK_FLAGS         = "C:\Program Files
(x86)\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

###############################################################################
#
#
# Microsoft Device Driver Kit 3790.1830 (IA-32, X64, Itanium, with Link Time Code
Generation)
# And Intel ACPI Compiler
#
###############################################################################
#
#   DDK3790  - Microsoft Windows DDK 3790.1830
#   ASL  - Intel ACPI Source Language Compiler (iasl.exe)
*_DDK3790_*_*_FAMILY        = MSFT

*_DDK3790_*_MAKE_PATH       = DEF(WINDDK_BIN32)\nmake.exe
*_DDK3790_*_MAKE_FLAGS          = /nologo
*_DDK3790_*_RC_PATH         = DEF(WINDDK_BIN32)\rc.exe

*_DDK3790_*_PP_FLAGS            = /nologo /E /TC /FIAutoGen.h
*_DDK3790_*_APP_FLAGS           = /nologo /E /TC
*_DDK3790_*_SLINK_FLAGS         = /nologo /LTCG
*_DDK3790_*_VFRPP_FLAGS         = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

##################
# ASL definitions
##################
*_DDK3790_*_ASL_PATH        = DEF(WIN_IASL_BIN)
*_DDK3790_*_ASL_FLAGS       =
*_DDK3790_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
```

```
*_DDK3790_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_DDK3790_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_DDK3790_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_DDK3790_IA32_CC_PATH      = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_SLINK_PATH   = DEF(WINDDK_BIN32)\lib.exe
*_DDK3790_IA32_DLINK_PATH   = DEF(WINDDK_BIN32)\link.exe
*_DDK3790_IA32_ASMLINK_PATH = DEF(WINDDK_BIN16)\link16.exe
*_DDK3790_IA32_PP_PATH      = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_VFRPP_PATH   = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_APP_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_ASM_PATH     = DEF(WINDDK_BIN32)\ml.exe
*_DDK3790_IA32_ASLCC_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_ASLPP_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790_IA32_ASLDLINK_PATH  = DEF(WINDDK_BIN32)\link.exe

  DEBUG_DDK3790_IA32_CC_FLAGS    = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE /O1ib2
/GL /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_DDK3790_IA32_CC_FLAGS    = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE /O1ib2
/GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_DDK3790_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /coff /Cx /Zd /Zi
RELEASE_DDK3790_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /coff /Cx /Zd
  DEBUG_DDK3790_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_DDK3790_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
    *_DDK3790_IA32_ASMLINK_FLAGS = /nologo /tiny

###################
# x64 definitions
###################
*_DDK3790_X64_CC_PATH      = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_SLINK_PATH   = DEF(WINDDK_BINX64)\lib.exe
*_DDK3790_X64_DLINK_PATH   = DEF(WINDDK_BINX64)\link.exe
*_DDK3790_X64_PP_PATH      = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_VFRPP_PATH   = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_APP_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_ASM_PATH     = DEF(WINDDK_BINX64)\ml64.exe
*_DDK3790_X64_ASLCC_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_ASLPP_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790_X64_ASLDLINK_PATH  = DEF(WINDDK_BINX64)\link.exe

  DEBUG_DDK3790_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_DDK3790_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE /
O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_DDK3790_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_DDK3790_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_DDK3790_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_DDK3790_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_DDK3790_IPF_CC_PATH      = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_SLINK_PATH   = DEF(WINDDK_BIN64)\lib.exe
*_DDK3790_IPF_DLINK_PATH   = DEF(WINDDK_BIN64)\link.exe
*_DDK3790_IPF_PP_PATH      = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_VFRPP_PATH   = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_APP_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_ASM_PATH     = DEF(WINDDK_BIN64)\ias.exe
*_DDK3790_IPF_ASLCC_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_ASLPP_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790_IPF_ASLDLINK_PATH  = DEF(WINDDK_BIN64)\link.exe


  DEBUG_DDK3790_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os /
GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_DDK3790_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /Os /
GL /FIAutoGen.h /QIPF_fr32
  DEBUG_DDK3790_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d debug
RELEASE_DDK3790_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_DDK3790_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF,ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /PDB:$(DEST_DIR_DEBUG)/
$(BASE_NAME).pdb /DEBUG
RELEASE_DDK3790_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF.ICF /
IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
```

```
DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /PDB:$(DEST_DIR_DEBUG)/
$(BASE_NAME).pdb

###################
# EBC definitions
###################
*_DDK3790_EBC_*_FAMILY      = INTEL

*_DDK3790_EBC_PP_PATH       = DEF(EBC_BIN)\iec.exe
*_DDK3790_EBC_CC_PATH       = DEF(EBC_BIN)\iec.exe
*_DDK3790_EBC_DLINK_PATH    = DEF(EBC_BIN)\link.exe
*_DDK3790_EBC_SLINK_PATH    = DEF(EBC_BIN)\link.exe
*_DDK3790_EBC_VFRPP_PATH    = DEF(EBC_BIN)\iec.exe

*_DDK3790_EBC_CC_FLAGS           = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_DDK3790_EBC_SLINK_FLAGS        = /lib /NOLOGO /MACHINE:EBC
*_DDK3790_EBC_DLINK_FLAGS        = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER

##############################################################################
#
#
# Microsoft Device Driver Kit 3790.1830 (IA-32, X64, Itanium, with Link Time Code
Generation)
# And Microsoft ACPI Compiler
#
##############################################################################
#
#   DDK3790xASL  - Microsoft Windows DDK 3790.1830
#   ASL          - Microsoft ACPI Source Language Compiler (asl.exe)
*_DDK3790xASL_*_*_FAMILY        = MSFT

*_DDK3790xASL_*_MAKE_PATH       = DEF(WINDDK_BIN32)\nmake.exe
*_DDK3790xASL_*_MAKE_FLAGS           = /nologo
*_DDK3790xASL_*_RC_PATH         = DEF(WINDDK_BIN32)\rc.exe

*_DDK3790xASL_*_PP_FLAGS             = /nologo /E /TC /FIAutoGen.h
*_DDK3790xASL_*_APP_FLAGS            = /nologo /E /TC
*_DDK3790xASL_*_SLINK_FLAGS          = /nologo /LTCG
*_DDK3790xASL_*_VFRPP_FLAGS          = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
*_DDK3790xASL_*_ASL_PATH        = DEF(WIN_ASL_BIN)
*_DDK3790xASL_*_ASL_FLAGS       =
*_DDK3790xASL_*_ASL_OUTFLAGS    = DEF(MS_ASL_OUTFLAGS)
*_DDK3790xASL_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_DDK3790xASL_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_DDK3790xASL_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

###################
```

```
# IA32 definitions
###################
*_DDK3790xASL_IA32_CC_PATH      = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_SLINK_PATH   = DEF(WINDDK_BIN32)\lib.exe
*_DDK3790xASL_IA32_DLINK_PATH   = DEF(WINDDK_BIN32)\link.exe
*_DDK3790xASL_IA32_ASMLINK_PATH = DEF(WINDDK_BIN16)\link16.exe
*_DDK3790xASL_IA32_PP_PATH      = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_VFRPP_PATH   = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_APP_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_ASM_PATH     = DEF(WINDDK_BIN32)\ml.exe
*_DDK3790xASL_IA32_ASLCC_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_ASLPP_PATH     = DEF(WINDDK_BIN32)\cl.exe
*_DDK3790xASL_IA32_ASLDLINK_PATH  = DEF(WINDDK_BIN32)\link.exe

  DEBUG_DDK3790xASL_IA32_CC_FLAGS    = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_DDK3790xASL_IA32_CC_FLAGS    = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE /
O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_DDK3790xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /coff /Cx /Zd /Zi
RELEASE_DDK3790xASL_IA32_ASM_FLAGS   = /nologo /c /WX /W3 /coff /Cx /Zd
  DEBUG_DDK3790xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_DDK3790xASL_IA32_DLINK_FLAGS = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text
    *_DDK3790xASL_IA32_ASMLINK_FLAGS = /nologo /tiny

###################
# x64 definitions
###################
*_DDK3790xASL_X64_CC_PATH      = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_SLINK_PATH   = DEF(WINDDK_BINX64)\lib.exe
*_DDK3790xASL_X64_DLINK_PATH   = DEF(WINDDK_BINX64)\link.exe
*_DDK3790xASL_X64_PP_PATH      = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_VFRPP_PATH   = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_APP_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_ASM_PATH     = DEF(WINDDK_BINX64)\ml64.exe
*_DDK3790xASL_X64_ASLCC_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_ASLPP_PATH     = DEF(WINDDK_BINX64)\cl.exe
*_DDK3790xASL_X64_ASLDLINK_PATH  = DEF(WINDDK_BINX64)\link.exe

  DEBUG_DDK3790xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE
/O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_DDK3790xASL_X64_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /Gs8192 /D UNICODE
/O1ib2s /GL /Gy /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_DDK3790xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_DDK3790xASL_X64_ASM_FLAGS    = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_DDK3790xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
```

```
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
DEBUG
RELEASE_DDK3790xASL_X64_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /OPT:REF /
OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /Machine:AMD64 /LTCG /
DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /DRIVER /
MERGE:.data=.text /MERGE:.rdata=.text

###################
# IPF definitions
###################
*_DDK3790xASL_IPF_CC_PATH      = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_SLINK_PATH   = DEF(WINDDK_BIN64)\lib.exe
*_DDK3790xASL_IPF_DLINK_PATH   = DEF(WINDDK_BIN64)\link.exe
*_DDK3790xASL_IPF_PP_PATH      = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_VFRPP_PATH   = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_APP_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_ASM_PATH     = DEF(WINDDK_BIN64)\ias.exe
*_DDK3790xASL_IPF_ASLCC_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_ASLPP_PATH     = DEF(WINDDK_BIN64)\cl.exe
*_DDK3790xASL_IPF_ASLDLINK_PATH  = DEF(WINDDK_BIN64)\link.exe


  DEBUG_DDK3790xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /
Os /GL /FIAutoGen.h /QIPF_fr32 /Zi
RELEASE_DDK3790xASL_IPF_CC_FLAGS     = /nologo /c /WX /GS- /X /W4 /EHs-c- /GR- /Gy /
Os /GL /FIAutoGen.h /QIPF_fr32
  DEBUG_DDK3790xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4 -d debug
RELEASE_DDK3790xASL_IPF_ASM_FLAGS    = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_DDK3790xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF,ICF
/IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /PDB:$(DEST_DIR_DEBUG)/
$(BASE_NAME).pdb /DEBUG
RELEASE_DDK3790xASL_IPF_DLINK_FLAGS  = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF.ICF
/IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:IA64 /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /SAFESEH:NO /BASE:0 /
```

```
DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /PDB:$(DEST_DIR_DEBUG)/
$(BASE_NAME).pdb

###################
# EBC definitions
###################
*_DDK3790xASL_EBC_*_FAMILY      = INTEL

*_DDK3790xASL_EBC_PP_PATH       = DEF(EBC_BIN)\iec.exe
*_DDK3790xASL_EBC_CC_PATH       = DEF(EBC_BIN)\iec.exe
*_DDK3790xASL_EBC_DLINK_PATH    = DEF(EBC_BIN)\link.exe
*_DDK3790xASL_EBC_SLINK_PATH    = DEF(EBC_BIN)\link.exe
*_DDK3790xASL_EBC_VFRPP_PATH    = DEF(EBC_BIN)\iec.exe

*_DDK3790xASL_EBC_CC_FLAGS           = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_DDK3790xASL_EBC_SLINK_FLAGS        = /lib /NOLOGO /MACHINE:EBC
*_DDK3790xASL_EBC_DLINK_FLAGS        = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER


#############################################################################
#
# GCC Common
#############################################################################
#

*_*_*_OBJCOPY_PATH              = echo
*_*_*_OBJCOPY_FLAGS             = objcopy not needed for
*_*_*_SYMRENAME_PATH            = echo
*_*_*_SYMRENAME_FLAGS           = Symbol renaming not needed for

DEFINE GCC_ALL_CC_FLAGS           = -Os -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h
DEFINE GCC_IA32_CC_FLAGS          = DEF(GCC_ALL_CC_FLAGS) -m32 -malign-double -
freorder-blocks -freorder-blocks-and-partition -O2
DEFINE GCC_X64_CC_FLAGS          = DEF(GCC_ALL_CC_FLAGS) -mno-red-zone -Wno-address
DEFINE GCC_IPF_CC_FLAGS           = DEF(GCC_ALL_CC_FLAGS) -minline-int-divide-min-
latency
DEFINE GCC_IA32_X64_DLINK_COMMON   = -nostdlib -s --pie --gc-sections
DEFINE GCC_IA32_X64_ASLDLINK_FLAGS = DEF(GCC_IA32_X64_DLINK_COMMON) --entry
_ReferenceAcpiTable -u $(IMAGE_ENTRY_POINT)
DEFINE GCC_IA32_X64_DLINK_FLAGS    = DEF(GCC_IA32_X64_DLINK_COMMON) --entry
_$(IMAGE_ENTRY_POINT) --file-alignment 0x20 --section-alignment 0x20 -Map
$(DEST_DIR_DEBUG)/$(BASE_NAME).map
DEFINE GCC_IPF_DLINK_FLAGS        = -nostdlib -O2 --gc-sections --dll -static --
entry $(IMAGE_ENTRY_POINT) --undefined $(IMAGE_ENTRY_POINT) -Map $(DEST_DIR_DEBUG)/
$(BASE_NAME).map
DEFINE GCC_IPF_OBJCOPY_FLAGS      = -I elf64-ia64-little -O efi-bsdrv-ia64
DEFINE GCC_IPF_SYMRENAME_FLAGS    = --redefine-sym memcpy=CopyMem
DEFINE GCC_ASM_FLAGS              = -c -x assembler -imacros $(DEST_DIR_DEBUG)/
AutoGen.h
DEFINE GCC_PP_FLAGS               = -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
```

```
DEFINE GCC_VFRPP_FLAGS             = -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h

##############################################################################
#
#
# Unix GCC And Intel Linux ACPI Compiler
#
##############################################################################
#
#   UNIXGCC         - UNIX GCC
#   ASL             - Intel Linux ACPI Source Language Compiler (iasl)
*_UNIXGCC_*_*_FAMILY               = GCC
*_UNIXGCC_*_MAKE_PATH              = make

DEFINE UNIXGCC_DLINK_FLAGS_COMMON        = -nostdlib -s --pie
*_UNIXGCC_*_PP_FLAGS                     = -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_UNIXGCC_*_DLINK_FLAGS                  = DEF(UNIXGCC_DLINK_FLAGS_COMMON) --entry
_$(IMAGE_ENTRY_POINT) --file-alignment 0x20 --section-alignment 0x20 -Map
$(DEST_DIR_DEBUG)/$(BASE_NAME).map
*_UNIXGCC_*_ASM_FLAGS                    = -c -x assembler -imacros
$(DEST_DIR_DEBUG)/AutoGen.h
*_UNIXGCC_*_VFRPP_FLAGS                  = -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
*_UNIXGCC_*_ASL_PATH                     = DEF(UNIX_IASL_BIN)
*_UNIXGCC_*_ASL_FLAGS                    =
*_UNIXGCC_*_ASL_OUTFLAGS                 = DEF(IASL_OUTFLAGS)
*_UNIXGCC_*_ASLPP_FLAGS                  = -x c -E -P
*_UNIXGCC_*_ASLCC_FLAGS                  = -x c
*_UNIXGCC_*_ASLDLINK_FLAGS               = DEF(UNIXGCC_DLINK_FLAGS_COMMON) --entry
_ReferenceAcpiTable

###################
# IA32 definitions
###################
*_UNIXGCC_IA32_PP_PATH              = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_CC_PATH              = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_SLINK_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)ar
*_UNIXGCC_IA32_DLINK_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)ld
*_UNIXGCC_IA32_ASLPP_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_ASLCC_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_ASLDLINK_PATH        = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)ld
*_UNIXGCC_IA32_ASM_PATH             = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_VFRPP_PATH           = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)gcc
*_UNIXGCC_IA32_RC_PATH              = DEF(UNIXGCC_IA32_PETOOLS_PREFIX)windres

*_UNIXGCC_IA32_CC_FLAGS             = -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -m32 -malign-
double -freorder-blocks -freorder-blocks-and-partition
*_UNIXGCC_IA32_RC_FLAGS             = -J rc -O coff
```

```
##################
# X64 definitions
##################
*_UNIXGCC_X64_CC_PATH               = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_ASLCC_PATH            = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_SLINK_PATH            = DEF(UNIXGCC_X64_PETOOLS_PREFIX)ar
*_UNIXGCC_X64_DLINK_PATH            = DEF(UNIXGCC_X64_PETOOLS_PREFIX)ld
*_UNIXGCC_X64_ASLDLINK_PATH         = DEF(UNIXGCC_X64_PETOOLS_PREFIX)ld
*_UNIXGCC_X64_ASM_PATH              = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_PP_PATH               = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_ASLPP_PATH            = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_VFRPP_PATH            = DEF(UNIXGCC_X64_PETOOLS_PREFIX)gcc
*_UNIXGCC_X64_RC_PATH               = DEF(UNIXGCC_X64_PETOOLS_PREFIX)windres

*_UNIXGCC_X64_CC_FLAGS              = -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -mno-red-zone -
Wno-address
*_UNIXGCC_X64_RC_FLAGS              = -J rc -O coff

##################
# IPF definitions
##################
*_UNIXGCC_IPF_CC_PATH              = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_ASLCC_PATH           = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_SLINK_PATH           = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)ar
*_UNIXGCC_IPF_DLINK_PATH           = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)ld
*_UNIXGCC_IPF_ASLDLINK_PATH        = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)ld
*_UNIXGCC_IPF_ASM_PATH             = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_PP_PATH              = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_ASLPP_PATH           = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_VFRPP_PATH           = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)gcc
*_UNIXGCC_IPF_OBJCOPY_PATH         = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)objcopy
*_UNIXGCC_IPF_SYMRENAME_PATH       = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)objcopy
*_UNIXGCC_IPF_RC_PATH              = DEF(UNIXGCC_IPF_PETOOLS_PREFIX)objcopy

*_UNIXGCC_IPF_CC_FLAGS             = -Os -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -minline-int-
divide-min-latency
*_UNIXGCC_IPF_DLINK_FLAGS          = -nostdlib -O2 --gc-sections --dll -static --
entry $(IMAGE_ENTRY_POINT) --undefined $(IMAGE_ENTRY_POINT) -Map $(DEST_DIR_DEBUG)/
$(BASE_NAME).map
*_UNIXGCC_IPF_OBJCOPY_FLAGS        = -I elf64-ia64-little -O efi-bsdrv-ia64
*_UNIXGCC_IPF_SYMRENAME_FLAGS      = --redefine-sym memcpy=CopyMem
*_UNIXGCC_IPF_RC_FLAGS             = -I binary -O elf64-ia64 --rename-section
.data=.hii

#############################################################################
#
#
# Cygwin GCC And Intel ACPI Compiler
#
#############################################################################
#
#   CYGGCC        - CygWin GCC
```

```
#   ASL           - Intel ACPI Source Language Compiler (iasl.exe)
*_CYGGCC_*_*_FAMILY               = GCC

*_CYGGCC_*_*_DLL                  = DEF(CYGWIN_BIN)
*_CYGGCC_*_MAKE_PATH              = DEF(MS_VS_BIN)\nmake.exe

*_CYGGCC_*_MAKE_FLAGS                 = /nologo
*_CYGGCC_*_PP_FLAGS                   = -E -x assembler-with-cpp -include AutoGen.h
*_CYGGCC_*_DLINK_FLAGS                = -nostdlib -O2 --gc-sections --dll --export-
all-symbols --entry _$(IMAGE_ENTRY_POINT) --file-alignment 0x20 --section-alignment
0x20
*_CYGGCC_*_ASM_FLAGS                  = -c -x assembler -imacros AutoGen.h
*_CYGGCC_*_APP_FLAGS                  = -E -x assembler
*_CYGGCC_*_VFRPP_FLAGS                = -x c -E -P -DVFRCOMPILE --include
$(MODULE_NAME)StrDefs.h

##################
# ASL definitions
##################
*_CYGGCC_*_ASL_PATH                   = DEF(WIN_IASL_BIN)
*_CYGGCC_*_ASL_FLAGS                  =
*_CYGGCC_*_ASL_OUTFLAGS               = DEF(IASL_OUTFLAGS)
*_CYGGCC_*_ASLPP_FLAGS                = -x c -E -P
*_CYGGCC_*_ASLCC_FLAGS                = -x c
*_CYGGCC_*_ASLDLINK_FLAGS             = -nostdlib -s --pie --gc-sections --entry
_ReferenceAcpiTable -u $(IMAGE_ENTRY_POINT)

##################
# IA32 definitions
##################
*_CYGGCC_IA32_CC_PATH             = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_SLINK_PATH          = DEF(CYGWIN_BINIA32)ar
*_CYGGCC_IA32_DLINK_PATH          = DEF(CYGWIN_BINIA32)ld
*_CYGGCC_IA32_ASM_PATH            = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_PP_PATH             = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_APP_PATH            = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_VFRPP_PATH          = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_ASLCC_PATH          = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_ASLPP_PATH          = DEF(CYGWIN_BINIA32)gcc
*_CYGGCC_IA32_ASLDLINK_PATH       = DEF(CYGWIN_BINIA32)ld
*_CYGGCC_IA32_RC_PATH             = DEF(CYGWIN_BINIA32)windres

*_CYGGCC_IA32_CC_FLAGS            = -Os -fshort-wchar -fno-strict-aliasing -Wall -
Werror -c -include AutoGen.h
*_CYGGCC_IA32_RC_FLAGS            = -J rc -O coff

##################
# X64 definitions
##################
*_CYGGCC_X64_CC_PATH              = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_SLINK_PATH           = DEF(CYGWIN_BINX64)ar
*_CYGGCC_X64_DLINK_PATH           = DEF(CYGWIN_BINX64)ld
*_CYGGCC_X64_ASM_PATH             = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_PP_PATH              = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_APP_PATH             = DEF(CYGWIN_BINX64)gcc
```

```
*_CYGGCC_X64_VFRPP_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_ASLCC_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_ASLPP_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCC_X64_ASLDLINK_PATH       = DEF(CYGWIN_BINX64)ld
*_CYGGCC_X64_RC_PATH             = DEF(CYGWIN_BINX64)windres

*_CYGGCC_X64_CC_FLAGS            = -Os -fshort-wchar -fno-strict-aliasing -mno-red-
zone -Wall -Werror -c -include AutoGen.h -D_EFI_P64
*_CYGGCC_X64_RC_FLAGS            = -J rc -O coff

##################
# IPF definitions
##################
*_CYGGCC_IPF_CC_PATH             = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_SLINK_PATH          = DEF(CYGWIN_BINIPF)ar
*_CYGGCC_IPF_DLINK_PATH          = DEF(CYGWIN_BINIPF)ld
*_CYGGCC_IPF_ASLDLINK_PATH       = DEF(CYGWIN_BINIPF)ld
*_CYGGCC_IPF_ASM_PATH            = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_PP_PATH             = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_VFRPP_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_ASLCC_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_ASLPP_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCC_IPF_OBJCOPY_PATH        = DEF(CYGWIN_BINIPF)objcopy
*_CYGGCC_IPF_SYMRENAME_PATH      = DEF(CYGWIN_BINIPF)objcopy
*_CYGGCC_IPF_RC_PATH             = DEF(CYGWIN_BINIPF)objcopy

*_CYGGCC_IPF_CC_FLAGS             = -Os -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -minline-int-
divide-min-latency
*_CYGGCC_IPF_DLINK_FLAGS         = -nostdlib -O2 --gc-sections --dll -static --entry
$(IMAGE_ENTRY_POINT) --undefined $(IMAGE_ENTRY_POINT) -Map $(DEST_DIR_DEBUG)/
$(BASE_NAME).map
*_CYGGCC_IPF_OBJCOPY_FLAGS        = -I elf64-ia64-little -O efi-bsdrv-ia64
*_CYGGCC_IPF_SYMRENAME_FLAGS      = --redefine-sym memcpy=CopyMem
*_CYGGCC_IPF_RC_FLAGS             = -I binary -O elf64-ia64 --rename-section
.data=.hii

##################
# EBC definitions
##################
*_CYGGCC_EBC_*_FAMILY      = INTEL

*_CYGGCC_EBC_PP_PATH       = DEF(EBC_BIN)\iec.exe
*_CYGGCC_EBC_CC_PATH       = DEF(EBC_BIN)\iec.exe
*_CYGGCC_EBC_DLINK_PATH    = DEF(EBC_BIN)\link.exe
*_CYGGCC_EBC_SLINK_PATH    = DEF(EBC_BIN)\link.exe
*_CYGGCC_EBC_VFRPP_PATH    = DEF(EBC_BIN)\iec.exe
*_CYGGCC_EBC_RC_PATH       = DEF(MS_VS_BIN)\rc.exe

*_CYGGCC_EBC_CC_FLAGS           = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_CYGGCC_EBC_SLINK_FLAGS        = /lib /NOLOGO /MACHINE:EBC
*_CYGGCC_EBC_DLINK_FLAGS        = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER
```

```
################################################################################
#
#
# Cygwin GCC And Microsoft ACPI Compiler
#
################################################################################
#
#   CYGGCCxASL        - CygWin GCC
#   ASL           - Microsoft ACPI Source Language Compiler (asl.exe)
*_CYGGCCxASL_*_*_FAMILY              = GCC

*_CYGGCCxASL_*_*_DLL                 = DEF(CYGWIN_BIN)
*_CYGGCCxASL_*_MAKE_PATH             = DEF(MS_VS_BIN)\nmake.exe

*_CYGGCCxASL_*_MAKE_FLAGS                 = /nologo
*_CYGGCCxASL_*_PP_FLAGS                   = -E -x assembler-with-cpp -include
AutoGen.h
*_CYGGCCxASL_*_DLINK_FLAGS                = -nostdlib -O2 --gc-sections --dll --
export-all-symbols --entry _$(IMAGE_ENTRY_POINT) --file-alignment 0x20 --section-
alignment 0x20
*_CYGGCCxASL_*_ASM_FLAGS                  = -c -x assembler -imacros AutoGen.h
*_CYGGCCxASL_*_APP_FLAGS                  = -E -x assembler
*_CYGGCCxASL_*_VFRPP_FLAGS                = -x c -E -P -DVFRCOMPILE --include
$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
*_CYGGCCxASL_*_ASL_PATH        = DEF(WIN_ASL_BIN)
*_CYGGCCxASL_*_ASL_FLAGS       =
*_CYGGCCxASL_*_ASL_OUTFLAGS    = DEF(MS_ASL_OUTFLAGS)
*_CYGGCCxASL_*_ASLCC_FLAGS     = DEF(MSFT_ASLCC_FLAGS)
*_CYGGCCxASL_*_ASLPP_FLAGS     = DEF(MSFT_ASLPP_FLAGS)
*_CYGGCCxASL_*_ASLDLINK_FLAGS  = DEF(MSFT_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_CYGGCCxASL_IA32_CC_PATH            = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_SLINK_PATH         = DEF(CYGWIN_BINIA32)ar
*_CYGGCCxASL_IA32_DLINK_PATH         = DEF(CYGWIN_BINIA32)ld
*_CYGGCCxASL_IA32_ASM_PATH           = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_PP_PATH            = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_APP_PATH           = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_VFRPP_PATH         = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_ASLCC_PATH         = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_ASLPP_PATH         = DEF(CYGWIN_BINIA32)gcc
*_CYGGCCxASL_IA32_ASLDLINK_PATH      = DEF(CYGWIN_BINIA32)ld
*_CYGGCCxASL_IA32_RC_PATH            = DEF(CYGWIN_BINIA32)windres

*_CYGGCCxASL_IA32_CC_FLAGS          = -Os -fshort-wchar -fno-strict-aliasing -Wall
-Werror -c -include AutoGen.h
*_CYGGCCxASL_IA32_RC_FLAGS           = -J rc -O coff
```

```
##################
# X64 definitions
##################
*_CYGGCCxASL_X64_CC_PATH             = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_SLINK_PATH          = DEF(CYGWIN_BINX64)ar
*_CYGGCCxASL_X64_DLINK_PATH          = DEF(CYGWIN_BINX64)ld
*_CYGGCCxASL_X64_ASM_PATH            = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_PP_PATH             = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_APP_PATH            = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_VFRPP_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_ASLCC_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_ASLPP_PATH          = DEF(CYGWIN_BINX64)gcc
*_CYGGCCxASL_X64_ASLDLINK_PATH       = DEF(CYGWIN_BINX64)ld
*_CYGGCCxASL_X64_RC_PATH             = DEF(CYGWIN_BINX64)windres

*_CYGGCCxASL_X64_CC_FLAGS            = -Os -fshort-wchar -fno-strict-aliasing -mno-
red-zone -Wall -Werror -c -include AutoGen.h -D_EFI_P64
*_CYGGCCxASL_X64_RC_FLAGS            = -J rc -O coff

##################
# IPF definitions
##################
*_CYGGCCxASL_IPF_CC_PATH             = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_SLINK_PATH          = DEF(CYGWIN_BINIPF)ar
*_CYGGCCxASL_IPF_DLINK_PATH          = DEF(CYGWIN_BINIPF)ld
*_CYGGCCxASL_IPF_ASLDLINK_PATH       = DEF(CYGWIN_BINIPF)ld
*_CYGGCCxASL_IPF_ASM_PATH            = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_PP_PATH             = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_VFRPP_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_ASLCC_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_ASLPP_PATH          = DEF(CYGWIN_BINIPF)gcc
*_CYGGCCxASL_IPF_OBJCOPY_PATH        = DEF(CYGWIN_BINIPF)objcopy
*_CYGGCCxASL_IPF_SYMRENAME_PATH      = DEF(CYGWIN_BINIPF)objcopy
*_CYGGCCxASL_IPF_RC_PATH             = DEF(CYGWIN_BINIPF)objcopy

*_CYGGCCxASL_IPF_CC_FLAGS            = -Os -fshort-wchar -fno-strict-aliasing -Wall
-Werror -Wno-missing-braces -Wno-array-bounds -c -include AutoGen.h -minline-int-
divide-min-latency
*_CYGGCCxASL_IPF_DLINK_FLAGS          = -nostdlib -O2 --gc-sections --dll -static -
-entry $(IMAGE_ENTRY_POINT) --undefined $(IMAGE_ENTRY_POINT) -Map
$(DEST_DIR_DEBUG)/$(BASE_NAME).map
*_CYGGCCxASL_IPF_OBJCOPY_FLAGS        = -I elf64-ia64-little -O efi-bsdrv-ia64
*_CYGGCCxASL_IPF_SYMRENAME_FLAGS      = --redefine-sym memcpy=CopyMem
*_CYGGCCxASL_IPF_RC_FLAGS             = -I binary -O elf64-ia64 --rename-section
.data=.hii

##################
# EBC definitions
##################
*_CYGGCCxASL_EBC_*_FAMILY      = INTEL

*_CYGGCCxASL_EBC_PP_PATH       = DEF(EBC_BIN)\iec.exe
*_CYGGCCxASL_EBC_CC_PATH       = DEF(EBC_BIN)\iec.exe
*_CYGGCCxASL_EBC_DLINK_PATH    = DEF(EBC_BIN)\link.exe
*_CYGGCCxASL_EBC_SLINK_PATH    = DEF(EBC_BIN)\link.exe
```

```
*_CYGGCCxASL_EBC_VFRPP_PATH    = DEF(EBC_BIN)\iec.exe
*_CYGGCCxASL_EBC_RC_PATH       = DEF(MS_VS_BIN)\rc.exe

*_CYGGCCxASL_EBC_CC_FLAGS           = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_CYGGCCxASL_EBC_SLINK_FLAGS        = /lib /NOLOGO /MACHINE:EBC
*_CYGGCCxASL_EBC_DLINK_FLAGS        = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER

################################################################################
#
#
# Elf GCC - This configuration is used to compile on Linux boxes to produce elf
#           binaries.
#
################################################################################
#
#   ELFGCC           - Linux ELF GCC
*_ELFGCC_*_*_FAMILY                 = GCC
*_ELFGCC_*_MAKE_PATH                = make

*_ELFGCC_*_PP_FLAGS                   = -E -x assembler-with-cpp -include AutoGen.h
*_ELFGCC_*_VFRPP_FLAGS                 = -x c -E -P -DVFRCOMPILE --include
$(MODULE_NAME)StrDefs.h

###################
# ASL definitions
###################
*_ELFGCC_*_ASL_PATH                    = DEF(UNIX_IASL_BIN)
*_ELFGCC_*_ASL_FLAGS                   =
*_ELFGCC_*_ASL_OUTFLAGS                = DEF(IASL_OUTFLAGS)
*_ELFGCC_*_ASLPP_FLAGS                 = -x c -E -P
*_ELFGCC_*_ASLCC_FLAGS                 = -x c
*_ELFGCC_*_ASLDLINK_FLAGS              = DEF(UNIXGCC_DLINK_FLAGS_COMMON) --entry
_ReferenceAcpiTable

###################
# IA32 definitions
###################
*_ELFGCC_IA32_CC_PATH               = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IA32_SLINK_PATH            = DEF(ELFGCC_BIN)/ar
*_ELFGCC_IA32_DLINK_PATH            = DEF(ELFGCC_BIN)/ld
*_ELFGCC_IA32_ASM_PATH              = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IA32_PP_PATH               = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IA32_VFRPP_PATH            = DEF(ELFGCC_BIN)/gcc
*_ELFGCC32_IA32_ASLCC_PATH          = DEF(ELFGCC_BIN)/gcc
*_ELFGCC32_IA32_ASLPP_PATH          = DEF(ELFGCC_BIN)/gcc
*_ELFGCC32_IA32_ASLDLINK_PATH       = DEF(ELFGCC_BIN)/ld
*_ELFGCC_IA32_RC_PATH               = DEF(ELFGCC_BIN)/objcopy

*_ELFGCC_IA32_CC_FLAGS              = -m32 -g -fshort-wchar -fno-strict-aliasing -
Wall -malign-double -c -include $(DEST_DIR_DEBUG)/AutoGen.h -
DSTRING_ARRAY_NAME=$(BASE_NAME)Strings
*_ELFGCC_IA32_SLINK_FLAGS           =
```

```
*_ELFGCC_IA32_DLINK_FLAGS           = -melf_i386 -nostdlib --shared --entry
$(IMAGE_ENTRY_POINT) -u $(IMAGE_ENTRY_POINT)
#*_ELFGCC_IA32_DLINK_FLAGS         = -melf_i386 -nostdlib -n -q -Ttext 0x220 --entry
$(IMAGE_ENTRY_POINT) -u $(IMAGE_ENTRY_POINT)
*_ELFGCC_IA32_ASM_FLAGS             = -c -x assembler -imacros $(DEST_DIR_DEBUG)/
AutoGen.h
*_ELFGCC_IA32_PP_FLAGS              = -m32 -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_ELFGCC_IA32_VFRPP_FLAGS           = -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h
*_ELFGCC_IA32_RC_FLAGS              = -I binary -O elf32-i386 -B i386 --rename-
section .data=.hii

##################
# X64 definitions
##################
*_ELFGCC_X64_CC_PATH               = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_ASLCC_PATH            = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_SLINK_PATH            = DEF(ELFGCC_BIN)/ar
*_ELFGCC_X64_DLINK_PATH            = DEF(ELFGCC_BIN)/ld
*_ELFGCC_X64_ASLDLINK_PATH         = DEF(ELFGCC_BIN)/ld
*_ELFGCC_X64_ASM_PATH              = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_PP_PATH               = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_ASLPP_PATH            = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_VFRPP_PATH            = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_X64_RC_PATH               = DEF(ELFGCC_BIN)/objcopy

*_ELFGCC_X64_CC_FLAGS              = -Os -fshort-wchar -fno-strict-aliasing -Wall -
Werror -Wno-missing-braces -Wno-address -Wno-array-bounds -c -include AutoGen.h -
D_EFI_P64
*_ELFGCC_X64_DLINK_FLAGS          = -nostdlib --shared --entry $(IMAGE_ENTRY_POINT)
-u $(IMAGE_ENTRY_POINT)
*_ELFGCC_X64_SLINK_FLAGS           =
*_ELFGCC_X64_ASM_FLAGS             = -c -x assembler -imacros $(DEST_DIR_DEBUG)/
AutoGen.h
*_ELFGCC_X64_PP_FLAGS              = -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_ELFGCC_X64_VFRPP_FLAGS           = -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h
*_ELFGCC_X64_RC_FLAGS              = -I binary -O elf64-x86-64 -B i386 --rename-
section .data=.hii

##################
# IPF definitions
##################
*_ELFGCC_IPF_CC_PATH              = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_ASLCC_PATH           = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_SLINK_PATH           = DEF(ELFGCC_BIN)/ar
*_ELFGCC_IPF_DLINK_PATH           = DEF(ELFGCC_BIN)/ld
*_ELFGCC_IPF_ASLDLINK_PATH        = DEF(ELFGCC_BIN)/ld
*_ELFGCC_IPF_ASM_PATH             = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_PP_PATH              = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_ASLPP_PATH           = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_VFRPP_PATH           = DEF(ELFGCC_BIN)/gcc
*_ELFGCC_IPF_RC_PATH              = DEF(ELFGCC_BIN)/objcopy
```

```
*_ELFGCC_IPF_CC_FLAGS             = -Os -fshort-wchar -Wall -Werror -c -include
AutoGen.h -D_EFI_P64
*_ELFGCC_IPF_DLINK_FLAGS          = -nostdlib --shared --entry $(IMAGE_ENTRY_POINT)
-u $(IMAGE_ENTRY_POINT)
*_ELFGCC_IPF_SLINK_FLAGS          =
*_ELFGCC_IPF_ASM_FLAGS            = -c -x assembler -imacros $(DEST_DIR_DEBUG)/
AutoGen.h
*_ELFGCC_IPF_PP_FLAGS             = -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_ELFGCC_IPF_VFRPP_FLAGS          = -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h
*_ELFGCC_IPF_RC_FLAGS             = -I binary -O elf64-ia64-little -B ia64 --rename-
section .data=.hii

##############################################################################
#
#
# Intel(R) C++ Compiler Version 9.1
#
#   IPF  - Intel(R) C++ Compiler for Itanium(R)Version 9.1  Build 20060928 Package
ID: W_CC_C_9.1.032
#   ASL  - Intel ACPI Source Language Compiler
#
##############################################################################
#
#   ICC              - Intel C Compiler V9.1
*_ICC_*_*_FAMILY                    = INTEL

*_ICC_*_MAKE_PATH                   = DEF(MS_VS_BIN)\nmake.exe
*_ICC_*_RC_PATH                     = DEF(MS_VS_BIN)\rc.exe

*_ICC_*_MAKE_FLAGS                      = /nologo
*_ICC_*_VFRPP_FLAGS                     = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICC_*_APP_FLAGS                       = /nologo /E /TC
*_ICC_*_PP_FLAGS                        = /nologo /E /TC /FIAutoGen.h

###################
# ASL definitions
###################
*_ICC_*_ASL_PATH        = DEF(WIN_IASL_BIN)
*_ICC_*_ASL_FLAGS       =
*_ICC_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
*_ICC_*_ASLCC_FLAGS     = DEF(ICC_WIN_ASLCC_FLAGS)
*_ICC_*_ASLPP_FLAGS     = DEF(ICC_WIN_ASLPP_FLAGS)
*_ICC_*_ASLDLINK_FLAGS  = DEF(ICC_WIN_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_ICC_IA32_CC_PATH                  = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_SLINK_PATH               = DEF(ICC_BIN32)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICC_IA32_SLINK_DLL                = DEF(MS_VS_BIN)
```

```
*_ICC_IA32_DLINK_PATH              = DEF(ICC_BIN32)\xilink.exe
*_ICC_IA32_ASMLINK_PATH            = DEF(WINDDK_BIN16)\link16.exe
*_ICC_IA32_PP_PATH                 = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_VFRPP_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_APP_PATH                = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_ASM_PATH                = DEF(MS_VS_BIN)\ml.exe
*_ICC_IA32_ASM_DLL                 = DEF(MS_VS_DLL)
*_ICC_IA32_ASLCC_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_ASLPP_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICC_IA32_ASLDLINK_PATH           = DEF(ICC_BIN32)\xilink.exe

  DEBUG_ICC_IA32_CC_FLAGS              = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE
/O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF /Zi /Gm
RELEASE_ICC_IA32_CC_FLAGS              = /nologo /c /WX /W4 /Gy /Gs8192 /D UNICODE
/O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF
  DEBUG_ICC_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd /Zi
RELEASE_ICC_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd
      *_ICC_IA32_SLINK_FLAGS            = /nologo
  DEBUG_ICC_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /DEBUG
RELEASE_ICC_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICC_IA32_ASMLINK_FLAGS          = /nologo /tiny

###################
# X64 definitions
###################
*_ICC_X64_CC_PATH                  = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_SLINK_PATH               = DEF(ICC_BINX64)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICC_X64_SLINK_DLL                = DEF(MS_VS_BIN)
*_ICC_X64_DLINK_PATH               = DEF(ICC_BINX64)\xilink.exe
*_ICC_X64_PP_PATH                  = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_VFRPP_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_APP_PATH                 = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_ASM_PATH                 = DEF(WINDDK_BINX64)\ml64.exe
*_ICC_X64_ASM_DLL                  = DEF(MS_VS_DLL)
*_ICC_X64_ASLCC_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_ASLPP_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICC_X64_ASLDLINK_PATH            = DEF(ICC_BINX64)\xilink.exe

  DEBUG_ICC_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /Zi /Gm /EHs-c- /GF
RELEASE_ICC_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF
  DEBUG_ICC_X64_ASM_FLAGS              = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_ICC_X64_ASM_FLAGS              = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_ICC_X64_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /DEBUG
RELEASE_ICC_X64_DLINK_FLAGS              = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICC_X64_SLINK_FLAGS              = /nologo /LTCG


##################
# IPF definitions
##################
*_ICC_IPF_CC_PATH                  = DEF(ICC_BIN64)\icl.exe
# icl.exe needs cl.exe from Visual Studio
*_ICC_IPF_CC_DLL                   = DEF(MS_VS_BIN)
*_ICC_IPF_SLINK_PATH               = DEF(ICC_BIN64)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICC_IPF_SLINK_DLL                = DEF(MS_VS_BIN);DEF(MS_VS_DLL)
*_ICC_IPF_DLINK_PATH               = DEF(ICC_BIN64)\xilink.exe
*_ICC_IPF_PP_PATH                  = DEF(ICC_BIN64)\icl.exe
*_ICC_IPF_VFRPP_PATH               = DEF(ICC_BIN64)\icl.exe
*_ICC_IPF_APP_PATH                 = DEF(ICC_BIN64)\icl.exe
*_ICC_IPF_ASM_PATH                 = DEF(ICC_BIN64)\ias.exe
*_ICC_IPF_ASLCC_PATH               = DEF(ICC_BIN64)\icl.exe
*_ICC_IPF_ASLPP_PATH               = DEF(ICC_BIN64)\icl.exe
*_ICC_IPF_ASLDLINK_PATH            = DEF(ICC_BIN64)\xilink.exe

  DEBUG_ICC_IPF_CC_FLAGS               = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF /Zi
RELEASE_ICC_IPF_CC_FLAGS             = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF
  DEBUG_ICC_IPF_ASM_FLAGS              = -N us -X explicit -M ilp64 -N so -W3 -d
debug -F COFF32
RELEASE_ICC_IPF_ASM_FLAGS            = -N us -X explicit -M ilp64 -N so -W3 -F
COFF32
  DEBUG_ICC_IPF_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb /DEBUG
RELEASE_ICC_IPF_DLINK_FLAGS          = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb
      *_ICC_IPF_SLINK_FLAGS              = /nologo

##################
# EBC definitions
##################
*_ICC_EBC_*_FAMILY              = INTEL

*_ICC_EBC_MAKE_PATH             = DEF(MS_VS_BIN)\nmake.exe
*_ICC_EBC_PP_PATH               = DEF(EBC_BIN)\iec.exe
*_ICC_EBC_VFRPP_PATH            = DEF(EBC_BIN)\iec.exe
*_ICC_EBC_CC_PATH               = DEF(EBC_BIN)\iec.exe
*_ICC_EBC_SLINK_PATH            = DEF(EBC_BIN)\link.exe
*_ICC_EBC_DLINK_PATH            = DEF(EBC_BIN)\link.exe

*_ICC_EBC_MAKE_FLAGS                = /nologo
*_ICC_EBC_PP_FLAGS                  = /nologo /E /TC /FIAutoGen.h
*_ICC_EBC_CC_FLAGS                  = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_ICC_EBC_VFRPP_FLAGS               = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICC_EBC_SLINK_FLAGS               = /lib /NOLOGO /MACHINE:EBC
*_ICC_EBC_DLINK_FLAGS               = "C:\Program Files\Intel\EBC\Lib\EbcLib.lib" /
NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /ENTRY:$(IMAGE_ENTRY_POINT) /
SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP

#############################################################################
#
#
# Intel(R) C++ Compiler Version 9.1
#
#   IPF  - Intel(R) C++ Compiler for Itanium(R)Version 9.1  Build 20060928 Package
ID: W_CC_C_9.1.032
#   ASL  - Microsoft ACPI Source Language Compiler
#
#############################################################################
#
#   ICCxASL             - Intel C Compiler V9.1
*_ICCxASL_*_*_FAMILY                = INTEL

*_ICCxASL_*_MAKE_PATH               = DEF(MS_VS_BIN)\nmake.exe
*_ICCxASL_*_RC_PATH                 = DEF(MS_VS_BIN)\rc.exe

*_ICCxASL_*_MAKE_FLAGS                  = /nologo
*_ICCxASL_*_VFRPP_FLAGS                 = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCxASL_*_APP_FLAGS                   = /nologo /E /TC
*_ICCxASL_*_PP_FLAGS                    = /nologo /E /TC /FIAutoGen.h

##################
# ASL definitions
##################
*_ICCxASL_*_ASL_PATH        = DEF(WIN_IASL_BIN)
*_ICCxASL_*_ASL_FLAGS       =
```

```
*_ICCxASL_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
*_ICCxASL_*_ASLCC_FLAGS     = DEF(ICC_WIN_ASLCC_FLAGS)
*_ICCxASL_*_ASLPP_FLAGS     = DEF(ICC_WIN_ASLPP_FLAGS)
*_ICCxASL_*_ASLDLINK_FLAGS  = DEF(ICC_WIN_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_ICCxASL_IA32_CC_PATH                 = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_SLINK_PATH              = DEF(ICC_BIN32)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCxASL_IA32_SLINK_DLL               = DEF(MS_VS_BIN)
*_ICCxASL_IA32_DLINK_PATH              = DEF(ICC_BIN32)\xilink.exe
*_ICCxASL_IA32_ASMLINK_PATH            = DEF(WINDDK_BIN16)\link16.exe
*_ICCxASL_IA32_PP_PATH                 = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_VFRPP_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_APP_PATH                = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_ASM_PATH                = DEF(MS_VS_BIN)\ml.exe
*_ICCxASL_IA32_ASM_DLL                 = DEF(MS_VS_DLL)
*_ICCxASL_IA32_ASLCC_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_ASLPP_PATH              = DEF(ICC_BIN32)\icl.exe
*_ICCxASL_IA32_ASLDLINK_PATH           = DEF(ICC_BIN32)\xilink.exe

  DEBUG_ICCxASL_IA32_CC_FLAGS               = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF /Zi /Gm
RELEASE_ICCxASL_IA32_CC_FLAGS             = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF
  DEBUG_ICCxASL_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd /Zi
RELEASE_ICCxASL_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd
      *_ICCxASL_IA32_SLINK_FLAGS            = /nologo
  DEBUG_ICCxASL_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /DEBUG
RELEASE_ICCxASL_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
```

```
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCxASL_IA32_ASMLINK_FLAGS           = /nologo /tiny

###################
# X64 definitions
###################
*_ICCxASL_X64_CC_PATH                  = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_SLINK_PATH               = DEF(ICC_BINX64)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCxASL_X64_SLINK_DLL                = DEF(MS_VS_BIN)
*_ICCxASL_X64_DLINK_PATH               = DEF(ICC_BINX64)\xilink.exe
*_ICCxASL_X64_PP_PATH                  = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_VFRPP_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_APP_PATH                 = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_ASM_PATH                 = DEF(WINDDK_BINX64)\ml64.exe
*_ICCxASL_X64_ASM_DLL                  = DEF(MS_VS_DLL)
*_ICCxASL_X64_ASLCC_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_ASLPP_PATH               = DEF(ICC_BINX64)\icl.exe
*_ICCxASL_X64_ASLDLINK_PATH            = DEF(ICC_BINX64)\xilink.exe

  DEBUG_ICCxASL_X64_CC_FLAGS                = /nologo /c /WX /GS- /X /W4 /Gs8192 /
D UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /Zi /Gm /EHs-c- /GF
RELEASE_ICCxASL_X64_CC_FLAGS              = /nologo /c /WX /GS- /X /W4 /Gs8192 /
D UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF
  DEBUG_ICCxASL_X64_ASM_FLAGS               = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_ICCxASL_X64_ASM_FLAGS             = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_ICCxASL_X64_DLINK_FLAGS             = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /DEBUG
RELEASE_ICCxASL_X64_DLINK_FLAGS           = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCxASL_X64_SLINK_FLAGS              = /nologo /LTCG


##################
# IPF definitions
##################
*_ICCxASL_IPF_CC_PATH                   = DEF(ICC_BIN64)\icl.exe
# icl.exe needs cl.exe from Visual Studio
*_ICCxASL_IPF_CC_DLL                    = DEF(MS_VS_BIN)
*_ICCxASL_IPF_SLINK_PATH                = DEF(ICC_BIN64)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCxASL_IPF_SLINK_DLL                 = DEF(MS_VS_BIN);DEF(MS_VS_DLL)
*_ICCxASL_IPF_DLINK_PATH                = DEF(ICC_BIN64)\xilink.exe
*_ICCxASL_IPF_PP_PATH                   = DEF(ICC_BIN64)\icl.exe
*_ICCxASL_IPF_VFRPP_PATH                = DEF(ICC_BIN64)\icl.exe
*_ICCxASL_IPF_APP_PATH                  = DEF(ICC_BIN64)\icl.exe
*_ICCxASL_IPF_ASM_PATH                  = DEF(ICC_BIN64)\ias.exe
*_ICCxASL_IPF_ASLCC_PATH                = DEF(ICC_BIN64)\icl.exe
*_ICCxASL_IPF_ASLPP_PATH                = DEF(ICC_BIN64)\icl.exe
*_ICCxASL_IPF_ASLDLINK_PATH             = DEF(ICC_BIN64)\xilink.exe

  DEBUG_ICCxASL_IPF_CC_FLAGS                = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF /Zi
RELEASE_ICCxASL_IPF_CC_FLAGS                = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF
  DEBUG_ICCxASL_IPF_ASM_FLAGS               = -N us -X explicit -M ilp64 -N so -W3
-d debug -F COFF32
RELEASE_ICCxASL_IPF_ASM_FLAGS               = -N us -X explicit -M ilp64 -N so -W3
-F COFF32
  DEBUG_ICCxASL_IPF_DLINK_FLAGS             = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb /DEBUG
RELEASE_ICCxASL_IPF_DLINK_FLAGS             = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb
      *_ICCxASL_IPF_SLINK_FLAGS              = /nologo

##################
# EBC definitions
##################
*_ICCxASL_EBC_*_FAMILY              = INTEL

*_ICCxASL_EBC_MAKE_PATH             = DEF(MS_VS_BIN)\nmake.exe
*_ICCxASL_EBC_PP_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCxASL_EBC_VFRPP_PATH            = DEF(EBC_BIN)\iec.exe
*_ICCxASL_EBC_CC_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCxASL_EBC_SLINK_PATH            = DEF(EBC_BIN)\link.exe
*_ICCxASL_EBC_DLINK_PATH            = DEF(EBC_BIN)\link.exe

*_ICCxASL_EBC_MAKE_FLAGS                = /nologo
*_ICCxASL_EBC_PP_FLAGS                  = /nologo /E /TC /FIAutoGen.h
*_ICCxASL_EBC_CC_FLAGS                  = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_ICCxASL_EBC_VFRPP_FLAGS               = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCxASL_EBC_SLINK_FLAGS               = /lib /NOLOGO /MACHINE:EBC
*_ICCxASL_EBC_DLINK_FLAGS               = "C:\Program
Files\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP


##############################################################################
#
#
# Intel(R) C++ Compiler Version 9.1 x86 (32-bit on 64-bit OS)
#
#   ICCx86  - Intel(R) C++ Compiler for Itanium(R)Version 9.1  Build 20060928
Package ID: W_CC_C_9.1.032
#   ASL  - Intel ACPI Source Language Compiler
#
##############################################################################
#
#   ICCx86              - Intel C Compiler V9.1
*_ICCx86_*_*_FAMILY                 = INTEL

*_ICCx86_*_MAKE_PATH                = DEF(VS2005x86_BIN)\nmake.exe
*_ICCx86_*_RC_PATH                  = DEF(VS2005x86_BIN)\rc.exe

*_ICCx86_*_MAKE_FLAGS                   = /nologo
*_ICCx86_*_VFRPP_FLAGS                  = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCx86_*_APP_FLAGS                    = /nologo /E /TC
*_ICCx86_*_PP_FLAGS                     = /nologo /E /TC /FIAutoGen.h

##################
# ASL definitions
##################
*_ICCx86_*_ASL_PATH        = DEF(WIN_IASL_BIN)
```

```
*_ICCx86_*_ASL_FLAGS       =
*_ICCx86_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
*_ICCx86_*_ASLCC_FLAGS     = DEF(ICC_WIN_ASLCC_FLAGS)
*_ICCx86_*_ASLPP_FLAGS     = DEF(ICC_WIN_ASLPP_FLAGS)
*_ICCx86_*_ASLDLINK_FLAGS  = DEF(ICC_WIN_ASLDLINK_FLAGS)

##################
# IA32 definitions
##################
*_ICCx86_IA32_CC_PATH                 = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_SLINK_PATH              = DEF(ICC_BIN32x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86_IA32_SLINK_DLL               = DEF(VS2005x86_BIN)
*_ICCx86_IA32_DLINK_PATH              = DEF(ICC_BIN32x86)\xilink.exe
*_ICCx86_IA32_ASMLINK_PATH            = DEF(WINDDK_BIN16)\link16.exe
*_ICCx86_IA32_PP_PATH                 = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_VFRPP_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_APP_PATH                = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_ASM_PATH                = DEF(VS2005x86_BIN)\ml.exe
*_ICCx86_IA32_ASM_DLL                 = DEF(VS2005x86_DLL)
*_ICCx86_IA32_ASLCC_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_ASLPP_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86_IA32_ASLDLINK_PATH           = DEF(ICC_BIN32x86)\xilink.exe

  DEBUG_ICCx86_IA32_CC_FLAGS               = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF /Zi /Gm
RELEASE_ICCx86_IA32_CC_FLAGS             = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF
  DEBUG_ICCx86_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd /Zi
RELEASE_ICCx86_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd
      *_ICCx86_IA32_SLINK_FLAGS            = /nologo
  DEBUG_ICCx86_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /DEBUG
RELEASE_ICCx86_IA32_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
```

```
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCx86_IA32_ASMLINK_FLAGS           = /nologo /tiny

###################
# X64 definitions
###################
*_ICCx86_X64_CC_PATH                  = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_SLINK_PATH               = DEF(ICC_BINX64x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86_X64_SLINK_DLL                = DEF(VS2005x86_BIN)
*_ICCx86_X64_DLINK_PATH               = DEF(ICC_BINX64x86)\xilink.exe
*_ICCx86_X64_PP_PATH                  = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_VFRPP_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_APP_PATH                 = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_ASM_PATH                 = DEF(WINDDK_BINX64)\ml64.exe
*_ICCx86_X64_ASM_DLL                  = DEF(VS2005x86_DLL)
*_ICCx86_X64_ASLCC_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_ASLPP_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86_X64_ASLDLINK_PATH            = DEF(ICC_BINX64x86)\xilink.exe

  DEBUG_ICCx86_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /Zi /Gm /EHs-c- /GF
RELEASE_ICCx86_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192 /D
UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF
  DEBUG_ICCx86_X64_ASM_FLAGS               = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_ICCx86_X64_ASM_FLAGS               = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_ICCx86_X64_DLINK_FLAGS             = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /DEBUG
RELEASE_ICCx86_X64_DLINK_FLAGS             = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCx86_X64_SLINK_FLAGS              = /nologo /LTCG


##################
# IPF definitions
##################
*_ICCx86_IPF_CC_PATH                  = DEF(ICC_BIN64x86)\icl.exe
# icl.exe needs cl.exe from Visual Studio
*_ICCx86_IPF_CC_DLL                   = DEF(VS2005x86_BIN)
*_ICCx86_IPF_SLINK_PATH               = DEF(ICC_BIN64x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86_IPF_SLINK_DLL                = DEF(VS2005x86_BIN);DEF(VS2005x86_DLL)
*_ICCx86_IPF_DLINK_PATH               = DEF(ICC_BIN64x86)\xilink.exe
*_ICCx86_IPF_PP_PATH                  = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86_IPF_VFRPP_PATH               = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86_IPF_APP_PATH                 = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86_IPF_ASM_PATH                 = DEF(ICC_BIN64x86)\ias.exe
*_ICCx86_IPF_ASLCC_PATH               = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86_IPF_ASLPP_PATH               = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86_IPF_ASLDLINK_PATH            = DEF(ICC_BIN64x86)\xilink.exe

  DEBUG_ICCx86_IPF_CC_FLAGS               = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF /Zi
RELEASE_ICCx86_IPF_CC_FLAGS               = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF
  DEBUG_ICCx86_IPF_ASM_FLAGS              = -N us -X explicit -M ilp64 -N so -W3 -
d debug -F COFF32
RELEASE_ICCx86_IPF_ASM_FLAGS              = -N us -X explicit -M ilp64 -N so -W3 -
F COFF32
  DEBUG_ICCx86_IPF_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb /DEBUG
RELEASE_ICCx86_IPF_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb
      *_ICCx86_IPF_SLINK_FLAGS              = /nologo

##################
# EBC definitions
##################
*_ICCx86_EBC_*_FAMILY              = INTEL

*_ICCx86_EBC_MAKE_PATH             = DEF(VS2005x86_BIN)\nmake.exe
*_ICCx86_EBC_PP_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCx86_EBC_VFRPP_PATH            = DEF(EBC_BIN)\iec.exe
*_ICCx86_EBC_CC_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCx86_EBC_SLINK_PATH            = DEF(EBC_BIN)\link.exe
*_ICCx86_EBC_DLINK_PATH            = DEF(EBC_BIN)\link.exe

*_ICCx86_EBC_MAKE_FLAGS                = /nologo
*_ICCx86_EBC_PP_FLAGS                  = /nologo /E /TC /FIAutoGen.h
*_ICCx86_EBC_CC_FLAGS                  = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_ICCx86_EBC_VFRPP_FLAGS               = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCx86_EBC_SLINK_FLAGS               = /lib /NOLOGO /MACHINE:EBC
*_ICCx86_EBC_DLINK_FLAGS               = "C:\Program Files
(x86)\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP


###############################################################################
#
#
# Intel(R) C++ Compiler Version 9.1 x86 (32-bit on 64-bit OS)
#
#   ICCx86xASL  - Intel(R) C++ Compiler for Itanium(R)Version 9.1  Build 20060928
Package ID: W_CC_C_9.1.032
#   ASL  - Microsoft ACPI Source Language Compiler
#
###############################################################################
#
#   ICCx86xASL             - Intel C Compiler V9.1
*_ICCx86xASL_*_*_FAMILY                    = INTEL

*_ICCx86xASL_*_MAKE_PATH                   = DEF(VS2005x86_BIN)\nmake.exe
*_ICCx86xASL_*_RC_PATH                     = DEF(VS2005x86_BIN)\rc.exe

*_ICCx86xASL_*_MAKE_FLAGS                      = /nologo
*_ICCx86xASL_*_VFRPP_FLAGS                     = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCx86xASL_*_APP_FLAGS                       = /nologo /E /TC
*_ICCx86xASL_*_PP_FLAGS                        = /nologo /E /TC /FIAutoGen.h

##################
# ASL definitions
##################
*_ICCx86xASL_*_ASL_PATH        = DEF(WIN_ASL_BIN)
```

```
*_ICCx86xASL_*_ASL_FLAGS       =
*_ICCx86xASL_*_ASL_OUTFLAGS    = DEF(MS_ASL_OUTFLAGS)
*_ICCx86xASL_*_ASLCC_FLAGS     = DEF(ICC_WIN_ASLCC_FLAGS)
*_ICCx86xASL_*_ASLPP_FLAGS     = DEF(ICC_WIN_ASLPP_FLAGS)
*_ICCx86xASL_*_ASLDLINK_FLAGS  = DEF(ICC_WIN_ASLDLINK_FLAGS)

###################
# IA32 definitions
###################
*_ICCx86xASL_IA32_CC_PATH                 = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_SLINK_PATH              = DEF(ICC_BIN32x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86xASL_IA32_SLINK_DLL               = DEF(VS2005x86_BIN)
*_ICCx86xASL_IA32_DLINK_PATH              = DEF(ICC_BIN32x86)\xilink.exe
*_ICCx86xASL_IA32_ASMLINK_PATH            = DEF(WINDDK_BIN16)\link16.exe
*_ICCx86xASL_IA32_PP_PATH                 = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_VFRPP_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_APP_PATH                = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_ASM_PATH                = DEF(VS2005x86_BIN)\ml.exe
*_ICCx86xASL_IA32_ASM_DLL                 = DEF(VS2005x86_DLL)
*_ICCx86xASL_IA32_ASLCC_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_ASLPP_PATH              = DEF(ICC_BIN32x86)\icl.exe
*_ICCx86xASL_IA32_ASLDLINK_PATH           = DEF(ICC_BIN32x86)\xilink.exe

  DEBUG_ICCx86xASL_IA32_CC_FLAGS               = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF /Zi /Gm
RELEASE_ICCx86xASL_IA32_CC_FLAGS             = /nologo /c /WX /W4 /Gy /Gs8192 /D
UNICODE /O1ib2 /GL /DEFI_FIRMWARE_VENDOR=L\"INTEL\" /FIAutoGen.h /EHs-c- /GF
  DEBUG_ICCx86xASL_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd /Zi
RELEASE_ICCx86xASL_IA32_ASM_FLAGS              = /nologo /c /WX /W3 /Zd
      *_ICCx86xASL_IA32_SLINK_FLAGS            = /nologo
  DEBUG_ICCx86xASL_IA32_DLINK_FLAGS           = /NOLOGO /NODEFAULTLIB /IGNORE:4086
/OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:I386 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /
BASE:0 /DRIVER /DEBUG
RELEASE_ICCx86xASL_IA32_DLINK_FLAGS           = /NOLOGO /NODEFAULTLIB /IGNORE:4086
/OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
MACHINE:I386 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /
BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCx86xASL_IA32_ASMLINK_FLAGS           = /nologo /tiny

###################
# X64 definitions
###################
*_ICCx86xASL_X64_CC_PATH                  = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_SLINK_PATH               = DEF(ICC_BINX64x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86xASL_X64_SLINK_DLL                = DEF(VS2005x86_BIN)
*_ICCx86xASL_X64_DLINK_PATH               = DEF(ICC_BINX64x86)\xilink.exe
*_ICCx86xASL_X64_PP_PATH                  = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_VFRPP_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_APP_PATH                 = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_ASM_PATH                 = DEF(WINDDK_BINX64)\ml64.exe
*_ICCx86xASL_X64_ASM_DLL                  = DEF(VS2005x86_DLL)
*_ICCx86xASL_X64_ASLCC_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_ASLPP_PATH               = DEF(ICC_BINX64x86)\icl.exe
*_ICCx86xASL_X64_ASLDLINK_PATH            = DEF(ICC_BINX64x86)\xilink.exe

  DEBUG_ICCx86xASL_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192
/D UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /Zi /Gm /EHs-c- /GF
RELEASE_ICCx86xASL_X64_CC_FLAGS               = /nologo /c /WX /GS- /X /W4 /Gs8192
/D UNICODE /O1ib2s /GL /Gy /FI$(DEST_DIR_DEBUG)/AutoGen.h /EHs-c- /GF
  DEBUG_ICCx86xASL_X64_ASM_FLAGS               = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_ICCx86xASL_X64_ASM_FLAGS               = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_ICCx86xASL_X64_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086
/OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /DEBUG
RELEASE_ICCx86xASL_X64_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /IGNORE:4086
/OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text
      *_ICCx86xASL_X64_SLINK_FLAGS              = /nologo /LTCG


##################
# IPF definitions
##################
*_ICCx86xASL_IPF_CC_PATH                   = DEF(ICC_BIN64x86)\icl.exe
# icl.exe needs cl.exe from Visual Studio
*_ICCx86xASL_IPF_CC_DLL                    = DEF(VS2005x86_BIN)
*_ICCx86xASL_IPF_SLINK_PATH                = DEF(ICC_BIN64x86)\xilib.exe
# xilib.exe needs lib.exe from Visual Studio
*_ICCx86xASL_IPF_SLINK_DLL                 = DEF(VS2005x86_BIN);DEF(VS2005x86_DLL)
*_ICCx86xASL_IPF_DLINK_PATH                = DEF(ICC_BIN64x86)\xilink.exe
*_ICCx86xASL_IPF_PP_PATH                   = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86xASL_IPF_VFRPP_PATH                = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86xASL_IPF_APP_PATH                  = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86xASL_IPF_ASM_PATH                  = DEF(ICC_BIN64x86)\ias.exe
*_ICCx86xASL_IPF_ASLCC_PATH                = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86xASL_IPF_ASLPP_PATH                = DEF(ICC_BIN64x86)\icl.exe
*_ICCx86xASL_IPF_ASLDLINK_PATH             = DEF(ICC_BIN64x86)\xilink.exe

  DEBUG_ICCx86xASL_IPF_CC_FLAGS               = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF /Zi
RELEASE_ICCx86xASL_IPF_CC_FLAGS             = /nologo /c /WX /W4 /GX /Gy /Od /
FI$(DEST_DIR_DEBUG)/AutoGen.h /QIA64_fr32 /GF
  DEBUG_ICCx86xASL_IPF_ASM_FLAGS              = -N us -X explicit -M ilp64 -N so -
W3 -d debug -F COFF32
RELEASE_ICCx86xASL_IPF_ASM_FLAGS            = -N us -X explicit -M ilp64 -N so -
W3 -F COFF32
  DEBUG_ICCx86xASL_IPF_DLINK_FLAGS            = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb /DEBUG
RELEASE_ICCx86xASL_IPF_DLINK_FLAGS          = /NOLOGO /NODEFAULTLIB /LTCG /DLL /
OPT:REF,ICF /IGNORE:4086 /MAP /ALIGN:64 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEST_DIR_DEBUG)/$(BASE_NAME).pdb
      *_ICCx86xASL_IPF_SLINK_FLAGS              = /nologo

###################
# EBC definitions
###################
*_ICCx86xASL_EBC_*_FAMILY              = INTEL

*_ICCx86xASL_EBC_MAKE_PATH             = DEF(VS2005x86_BIN)\nmake.exe
*_ICCx86xASL_EBC_PP_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCx86xASL_EBC_VFRPP_PATH            = DEF(EBC_BIN)\iec.exe
*_ICCx86xASL_EBC_CC_PATH               = DEF(EBC_BIN)\iec.exe
*_ICCx86xASL_EBC_SLINK_PATH            = DEF(EBC_BIN)\link.exe
*_ICCx86xASL_EBC_DLINK_PATH            = DEF(EBC_BIN)\link.exe

*_ICCx86xASL_EBC_MAKE_FLAGS                = /nologo
*_ICCx86xASL_EBC_PP_FLAGS                  = /nologo /E /TC /FIAutoGen.h
*_ICCx86xASL_EBC_CC_FLAGS                  = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_ICCx86xASL_EBC_VFRPP_FLAGS               = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_ICCx86xASL_EBC_SLINK_FLAGS               = /lib /NOLOGO /MACHINE:EBC
*_ICCx86xASL_EBC_DLINK_FLAGS               = "C:\Program Files
(x86)\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP


##############################################################################
#
#
# MYTOOLS
#   IA32 - Microsoft Visual Studio .NET 2003 and Microsoft Windows DDK 3790.1830 for
16-bit linker
#   X64  - Microsoft Windows DDK 3790.1830
#   IPF  - Microsoft Windows DDK 3790.1830
#   EBC  - Intel EFI Byte Code Compiler
#
##############################################################################
#
#   MYTOOLS          - Settings compatible with previous versions of
tools_def.template
*_MYTOOLS_*_*_FAMILY               = MSFT

###################
# ASL definitions
###################
*_MYTOOLS_*_ASL_PATH        = DEF(WIN_IASL_BIN)
*_MYTOOLS_*_ASL_FLAGS       =
*_MYTOOLS_*_ASL_OUTFLAGS    = DEF(IASL_OUTFLAGS)
*_MYTOOLS_*_ASLCC_FLAGS     = DEF(ICC_WIN_ASLCC_FLAGS)
*_MYTOOLS_*_ASLPP_FLAGS     = DEF(ICC_WIN_ASLPP_FLAGS)
*_MYTOOLS_*_ASLDLINK_FLAGS  = DEF(ICC_WIN_ASLDLINK_FLAGS)
```

```
*_MYTOOLS_*_MAKE_FLAGS                  = /nologo
*_MYTOOLS_*_VFRPP_FLAGS                 = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_MYTOOLS_*_APP_FLAGS                   = /nologo /E /TC
*_MYTOOLS_*_PP_FLAGS                    = /nologo /E /TC /FIAutoGen.h
*_MYTOOLS_*_SLINK_FLAGS                 = /nologo /LTCG

##################
# IA32 definitions
##################
*_MYTOOLS_IA32_*_DLL               = DEF(VS2005_DLL)

*_MYTOOLS_IA32_MAKE_PATH           = DEF(VS2005_BIN)\nmake.exe
*_MYTOOLS_IA32_CC_PATH             = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_SLINK_PATH          = DEF(VS2005_BIN)\lib.exe
*_MYTOOLS_IA32_DLINK_PATH          = DEF(VS2005_BIN)\link.exe
*_MYTOOLS_IA32_ASMLINK_PATH        = DEF(WINDDK_BIN16)\link16.exe
*_MYTOOLS_IA32_PP_PATH             = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_VFRPP_PATH          = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_APP_PATH            = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_ASM_PATH            = DEF(VS2005_BIN)\ml.exe
*_MYTOOLS_IA32_ASLCC_PATH          = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_ASLPP_PATH          = DEF(VS2005_BIN)\cl.exe
*_MYTOOLS_IA32_ASLDLINK_PATH       = DEF(VS2005_BIN)\link.exe
*_MYTOOLS_IA32_RC_PATH             = DEF(VS2005_BIN)\rc.exe

  DEBUG_MYTOOLS_IA32_CC_FLAGS           = /nologo /c /WX /GS- /W4 /Gs8192 /Gy /D
UNICODE /O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_MYTOOLS_IA32_CC_FLAGS           = /nologo /c /WX /GS- /W4 /Gs8192 /Gy /D
UNICODE /O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_MYTOOLS_IA32_ASM_FLAGS          = /nologo /c /WX /W3 /coff /Cx /Zd /Zi
RELEASE_MYTOOLS_IA32_ASM_FLAGS          = /nologo /c /WX /W3 /coff /Cx /Zd
  DEBUG_MYTOOLS_IA32_DLINK_FLAGS        = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /DEBUG /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb
RELEASE_MYTOOLS_IA32_DLINK_FLAGS        = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /MACHINE:I386
```

```
/LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO /BASE:0 /
DRIVER /MERGE:.data=.text /MERGE:.rdata=.text /PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb
      *_MYTOOLS_IA32_ASMLINK_FLAGS       = /nologo /tiny

###################
# x64 definitions
###################
*_MYTOOLS_X64_*_DLL                = DEF(VS2005_DLL)

*_MYTOOLS_X64_MAKE_PATH            = DEF(VS2005_BIN)\nmake.exe
*_MYTOOLS_X64_CC_PATH              = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_SLINK_PATH           = DEF(VS2005_BINX64)\lib.exe
*_MYTOOLS_X64_DLINK_PATH           = DEF(VS2005_BINX64)\link.exe
*_MYTOOLS_X64_PP_PATH              = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_VFRPP_PATH           = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_APP_PATH             = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_ASM_PATH             = DEF(VS2005_BINX64)\ml64.exe
*_MYTOOLS_X64_ASLCC_PATH           = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_ASLPP_PATH           = DEF(VS2005_BINX64)\cl.exe
*_MYTOOLS_X64_ASLDLINK_PATH        = DEF(VS2005_BINX64)\link.exe
*_MYTOOLS_X64_RC_PATH              = DEF(VS2005_BIN)\rc.exe

  DEBUG_MYTOOLS_X64_CC_FLAGS             = /nologo /c /WX /GS- /W4 /Gs8192 /Gy /D
UNICODE /O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF /Zi /Gm
RELEASE_MYTOOLS_X64_CC_FLAGS           = /nologo /c /WX /GS- /W4 /Gs8192 /Gy /D
UNICODE /O1ib2 /GL /FIAutoGen.h /EHs-c- /GR- /GF
  DEBUG_MYTOOLS_X64_ASM_FLAGS            = /nologo /c /WX /W3 /Cx /Zd /Zi
RELEASE_MYTOOLS_X64_ASM_FLAGS          = /nologo /c /WX /W3 /Cx /Zd
  DEBUG_MYTOOLS_X64_DLINK_FLAGS          = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /DEBUG
RELEASE_MYTOOLS_X64_DLINK_FLAGS          = /NOLOGO /NODEFAULTLIB /IGNORE:4086 /
OPT:REF /OPT:ICF=10 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
```

```
Machine:AMD64 /LTCG /DLL /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:CONSOLE /SAFESEH:NO
/BASE:0 /DRIVER /MERGE:.data=.text /MERGE:.rdata=.text

##################
# IPF definitions
##################
*_MYTOOLS_IPF_MAKE_PATH            = DEF(WINDDK_BIN32)\nmake.exe
*_MYTOOLS_IPF_CC_PATH              = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_SLINK_PATH           = DEF(WINDDK_BIN64)\lib.exe
*_MYTOOLS_IPF_DLINK_PATH           = DEF(WINDDK_BIN64)\link.exe
*_MYTOOLS_IPF_PP_PATH              = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_VFRPP_PATH           = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_APP_PATH             = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_ASM_PATH             = DEF(WINDDK_BIN64)\ias.exe
*_MYTOOLS_IPF_ASLCC_PATH           = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_ASLPP_PATH           = DEF(WINDDK_BIN64)\cl.exe
*_MYTOOLS_IPF_ASLDLINK_PATH        = DEF(WINDDK_BIN64)\link.exe
*_MYTOOLS_IPF_RC_PATH              = DEF(WINDDK_BIN32)\rc.exe

      *_MYTOOLS_IPF_ASM_OUTPUT           = "-o "
  DEBUG_MYTOOLS_IPF_CC_FLAGS             = /nologo /c /WX /GS- /X /W4 /Gy /Ox /GL /
FIAutoGen.h /EHs-c- /GR- /GF /Zx /QIPF_fr32 /Zi
RELEASE_MYTOOLS_IPF_CC_FLAGS           = /nologo /c /WX /GS- /X /W4 /Gy /Ox /GL /
FIAutoGen.h /EHs-c- /GR- /GF /Zx /QIPF_fr32
  DEBUG_MYTOOLS_IPF_ASM_FLAGS            = -N us -X explicit -M ilp64 -N so -W4 -d
debug
RELEASE_MYTOOLS_IPF_ASM_FLAGS          = -N us -X explicit -M ilp64 -N so -W4
  DEBUG_MYTOOLS_IPF_DLINK_FLAGS         = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF
/OPT:ICF=10 /IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb /DEBUG
RELEASE_MYTOOLS_IPF_DLINK_FLAGS        = /NOLOGO /NODEFAULTLIB /LTCG /DLL /OPT:REF
/OPT:ICF=10 /IGNORE:4086 /MAP /ALIGN:32 /SECTION:.xdata,D /SECTION:.pdata,D /
MACHINE:IA64 /ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /
```

```
SAFESEH:NO /BASE:0 /DRIVER /MAP:$(DEST_DIR_DEBUG)/$(BASE_NAME).map /
PDB:$(DEBUG_DIR)/$(BASE_NAME).pdb
      *_MYTOOLS_IPF_SLINK_FLAGS          = /nologo /LTCG

###################
# EBC definitions
###################
*_MYTOOLS_EBC_*_FAMILY              = INTEL

*_MYTOOLS_EBC_MAKE_PATH             = DEF(VS2005_BIN)\nmake.exe
*_MYTOOLS_EBC_PP_PATH               = DEF(EBC_BIN)\iec.exe
*_MYTOOLS_EBC_VFRPP_PATH            = DEF(EBC_BIN)\iec.exe
*_MYTOOLS_EBC_CC_PATH               = DEF(EBC_BIN)\iec.exe
*_MYTOOLS_EBC_SLINK_PATH            = DEF(EBC_BIN)\link.exe
*_MYTOOLS_EBC_DLINK_PATH            = DEF(EBC_BIN)\link.exe
*_MYTOOLS_EBC_RC_PATH               = DEF(VS2005_BIN)\rc.exe

*_MYTOOLS_EBC_MAKE_FLAGS                = /nologo
*_MYTOOLS_EBC_PP_FLAGS                  = /nologo /E /TC /FIAutoGen.h
*_MYTOOLS_EBC_CC_FLAGS                  = /nologo /c /WX /W3 /FIAutoGen.h /
D$(MODULE_ENTRY_POINT)=$(ARCH_ENTRY_POINT)
*_MYTOOLS_EBC_VFRPP_FLAGS               = /nologo /E /TC /DVFRCOMPILE /
FI$(MODULE_NAME)StrDefs.h
*_MYTOOLS_EBC_SLINK_FLAGS               = /lib /NOLOGO /MACHINE:EBC
*_MYTOOLS_EBC_DLINK_FLAGS               = "C:\Program
Files\Intel\EBC\Lib\EbcLib.lib" /NOLOGO /NODEFAULTLIB /MACHINE:EBC /OPT:REF /
ENTRY:$(IMAGE_ENTRY_POINT) /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER /MAP


##############################################################################
#
#
# Xcode Support for building on Mac OS X (Snow Leopard)
#
##############################################################################
#
#   XCODE32         - Xcode 3.2 Tools (Snow Leopard)
*_XCODE32_*_*_FAMILY            = GCC
*_XCODE32_*_*_BUILDRULEFAMILY   = XCODE


*_XCODE32_*_ASL_PATH          = /usr/bin/iasl

*_XCODE32_*_MAKE_PATH        = make

# This tool needs to be installed seperatly from Xcode 3.2
*_XCODE32_*_MTOC_PATH        = /Developer/usr/bin/mtoc
*_XCODE32_*_DSYMUTIL_PATH    = /usr/bin/dsymutil

  DEBUG_XCODE32_*_MTOC_FLAGS = -align 0x20 -d $(DEBUG_DIR)/$(MODULE_NAME).dll
RELEASE_XCODE32_*_MTOC_FLAGS = -align 0x20

###################
# IA32 definitions
###################
```

```
*_XCODE32_IA32_CC_PATH      = gcc
*_XCODE32_IA32_SLINK_PATH   = libtool
*_XCODE32_IA32_DLINK_PATH   = ld
*_XCODE32_IA32_ASM_PATH     = as
*_XCODE32_IA32_PP_PATH      = gcc
*_XCODE32_IA32_VFRPP_PATH   = gcc
*_XCODE32_IA32_ASL_PATH     = iasl
*_XCODE32_IA32_ASLCC_PATH   = gcc
*_XCODE32_IA32_ASLPP_PATH   = gcc
*_XCODE32_IA32_ASLDLINK_PATH  = ld

  DEBUG_XCODE32_IA32_DLINK_FLAGS      = -arch i386 -u _$(IMAGE_ENTRY_POINT) -e
_$(IMAGE_ENTRY_POINT) -preload -segalign 0x20 -slow_stubs -pie -all_load -dead_strip
-seg1addr 0x240 -read_only_relocs suppress -map $(DEST_DIR_DEBUG)/$(BASE_NAME).map
RELEASE_XCODE32_IA32_DLINK_FLAGS      = -arch i386 -u _$(IMAGE_ENTRY_POINT) -e
_$(IMAGE_ENTRY_POINT) -preload -segalign 0x20 -slow_stubs -pie -all_load -dead_strip
-seg1addr 0x220 -read_only_relocs suppress -map $(DEST_DIR_DEBUG)/$(BASE_NAME).map
*_XCODE32_IA32_SLINK_FLAGS      = -static -o
  DEBUG_XCODE32_IA32_ASM_FLAGS  = -arch i386 -g
RELEASE_XCODE32_IA32_ASM_FLAGS  = -arch i386
*_XCODE32_IA32_PP_FLAGS         = -arch i386 -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_XCODE32_IA32_VFRPP_FLAGS      = -arch i386 -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h
  DEBUG_XCODE32_IA32_CC_FLAGS   = -arch i386 -save-temps -g -O0 -combine -mms-
bitfields  -fshort-wchar -fno-strict-aliasing -Wall -Werror -Wno-missing-braces  -c
-include AutoGen.h -mdynamic-no-pic -fno-stack-protector
RELEASE_XCODE32_IA32_CC_FLAGS   = -arch i386 -Oz -combine -mms-bitfields  -fshort-
wchar -fno-strict-aliasing -Wall -Werror -Wno-missing-braces -fomit-frame-pointer -
c -include AutoGen.h -mdynamic-no-pic -fno-stack-protector

*_XCODE32_IA32_ASLCC_FLAGS      = -arch i386 -x c -save-temps -g -O0 -mms-bitfields
-fshort-wchar -fno-strict-aliasing -Wall -Werror -Wno-missing-braces -c -include
AutoGen.h -mdynamic-no-pic
*_XCODE32_IA32_ASLDLINK_FLAGS   = -arch i386 -e _main -preload -segalign 0x20 -
slow_stubs -pie -seg1addr 0x220 -read_only_relocs suppress -map $(DEST_DIR_DEBUG)/
$(BASE_NAME).map
*_XCODE32_IA32_ASLPP_FLAGS      = -arch i386 -x c -E
*_XCODE32_IA32_ASL_FLAGS        =

###################
# X64 definitions - still a work in progress. This tool chain does not produce
# the correct ABI, it is just used to compile the code....
###################
*_XCODE32_X64_CC_PATH       = gcc-4.2
*_XCODE32_X64_SLINK_PATH    = libtool
*_XCODE32_X64_DLINK_PATH    = ld
*_XCODE32_X64_ASM_PATH      = as
*_XCODE32_X64_PP_PATH       = gcc-4.2
*_XCODE32_X64_VFRPP_PATH    = gcc-4.2
*_XCODE32_X64_ASL_PATH     = iasl
*_XCODE32_X64_ASLCC_PATH   = gcc
*_XCODE32_X64_ASLPP_PATH   = gcc
*_XCODE32_X64_ASLDLINK_PATH  = ld
```

```
*_XCODE32_X64_DLINK_FLAGS      = -arch x86_64 -u _$(IMAGE_ENTRY_POINT) -e
_$(IMAGE_ENTRY_POINT) -preload -segalign 0x20 -slow_stubs -pie -seg1addr 0x240 -
read_only_relocs suppress -map $(DEST_DIR_DEBUG)/$(BASE_NAME).map
*_XCODE32_X64_SLINK_FLAGS      = -static -o

  DEBUG_XCODE32_X64_ASM_FLAGS  = -arch x86_64 -g
RELEASE_XCODE32_X64_ASM_FLAGS  = -arch x86_64
*_XCODE32_X64_PP_FLAGS         = -arch x86_64 -E -x assembler-with-cpp -include
$(DEST_DIR_DEBUG)/AutoGen.h
*_XCODE32_X64_VFRPP_FLAGS      = -arch x86_64 -x c -E -P -DVFRCOMPILE --include
$(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h

  DEBUG_XCODE32_X64_CC_FLAGS   = -arch x86_64 -save-temps -g -O0 -combine -mms-
bitfields -fshort-wchar -fno-strict-aliasing -Wall -Werror -Wno-missing-braces -Wno-
address  -fomit-frame-pointer -static  -c -include AutoGen.h -fno-stack-protector
RELEASE_XCODE32_X64_CC_FLAGS   = -arch x86_64 -Oz -combine -mms-bitfields -fshort-
wchar -fno-strict-aliasing -Wall -Werror -Wno-missing-braces -Wno-address  -fomit-
frame-pointer -static  -c -include AutoGen.h -fno-stack-protector

###################
# ARM definitions - still a work in progress (Assumes iPhone SDK installed)
###################

*_XCODE32_ARM_ARCHCC_FLAGS    = -arch armv6 -march=armv6 -mthumb
*_XCODE32_ARM_ARCHASM_FLAGS   = -arch armv6
*_XCODE32_ARM_ARCHDLINK_FLAGS = -arch armv6
*_XCODE32_ARM_PLATFORM_FLAGS  =
*_XCODE32_ARM_INTRINSIC_FLAGS = CompilerIntrinsicsLib.lib

*_XCODE32_ARM_CC_PATH       = DEF(IPHONE_TOOLS)/usr/bin/gcc
*_XCODE32_ARM_SLINK_PATH    = DEF(IPHONE_TOOLS)/usr/bin/libtool
*_XCODE32_ARM_DLINK_PATH    = DEF(IPHONE_TOOLS)/usr/bin/ld
*_XCODE32_ARM_ASM_PATH      = DEF(IPHONE_TOOLS)/usr/bin/as
*_XCODE32_ARM_PP_PATH       = DEF(IPHONE_TOOLS)/usr/bin/gcc
*_XCODE32_ARM_VFRPP_PATH    = DEF(IPHONE_TOOLS)/usr/bin/gcc

  DEBUG_XCODE32_ARM_DLINK_FLAGS      = $(ARCHDLINK_FLAGS) -u _$(IMAGE_ENTRY_POINT) -
e _$(IMAGE_ENTRY_POINT) -preload -segalign 0x20 -slow_stubs -pie -all_load -
dead_strip -seg1addr 0x220 -read_only_relocs suppress -map $(DEST_DIR_DEBUG)/
$(BASE_NAME).map ENV(EDK_TOOLS_PATH)/Bin/Darwin-i386/ARM/DEBUG_XCODE32/
$(INTRINSIC_FLAGS)
RELEASE_XCODE32_ARM_DLINK_FLAGS      = $(ARCHDLINK_FLAGS) -u _$(IMAGE_ENTRY_POINT) -
e _$(IMAGE_ENTRY_POINT) -preload -segalign 0x20 -slow_stubs -pie -all_load -
dead_strip -seg1addr 0x220 -read_only_relocs suppress -map $(DEST_DIR_DEBUG)/
```

```
$(BASE_NAME).map ENV(EDK_TOOLS_PATH)/Bin/Darwin-i386/ARM/RELEASE_XCODE32/
$(INTRINSIC_FLAGS)

*_XCODE32_ARM_SLINK_FLAGS      = -static -o

  DEBUG_XCODE32_ARM_ASM_FLAGS  = $(ARCHASM_FLAGS) -g
RELEASE_XCODE32_ARM_ASM_FLAGS  = $(ARCHASM_FLAGS)
*_XCODE32_ARM_PP_FLAGS         = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -E -x assembler-
with-cpp -include $(DEST_DIR_DEBUG)/AutoGen.h
*_XCODE32_ARM_VFRPP_FLAGS      = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -x c -E -P -
DVFRCOMPILE --include $(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h

  DEBUG_XCODE32_ARM_CC_FLAGS   = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -mthumb-interwork
-g -Oz -mabi=aapcs -mapcs -fno-short-enums  -save-temps -combine  -fshort-wchar -
fno-strict-aliasing -Wall -Werror -Wno-missing-braces -fomit-frame-pointer -c -
include AutoGen.h -mdynamic-no-pic -fno-stack-protector
RELEASE_XCODE32_ARM_CC_FLAGS   = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -mthumb-interwork
-Oz    -mabi=aapcs -mapcs -fno-short-enums  -save-temps -combine -fshort-wchar -fno-
strict-aliasing -Wall -Werror -Wno-missing-braces -fomit-frame-pointer -c -include
AutoGen.h -mdynamic-no-pic -fno-stack-protector


##############################################################################
#
#
# ARM Tools
#
##############################################################################
#
#   RVCT31         - Tools from ARM

*_RVCT31_*_*_FAMILY          = RVCT

#
# Use default values, or override in DSC file
#
*_RVCT31_ARM_ARCHCC_FLAGS    = --thumb --cpu ARM1176JZF-S
*_RVCT31_ARM_ARCHASM_FLAGS   = --cpu ARM1176JZF-S
*_RVCT31_ARM_ARCHDLINK_FLAGS =
*_RVCT31_ARM_PLATFORM_FLAGS  =
*_RVCT31_ARM_INTRINSIC_FLAGS = CompilerIntrinsicsLib.lib

  DEBUG_RVCT31_ARM_DLINK_FLAGS    = $(ARCHDLINK_FLAGS) --entry $(IMAGE_ENTRY_POINT)
--ro-base 0 --no_scanlib --reloc --no_exceptions --datacompressor off --strict --
symbols --map --list $(DEST_DIR_DEBUG)/$(BASE_NAME).map ENV(EDK_TOOLS_PATH)/Bin/
Win32/ARM/DEBUG_RVCT31/$(INTRINSIC_FLAGS)
RELEASE_RVCT31_ARM_DLINK_FLAGS    = $(ARCHDLINK_FLAGS) --entry $(IMAGE_ENTRY_POINT)
--ro-base 0 --no_scanlib --reloc --no_exceptions --datacompressor off --strict --
```

```
symbols --map --list $(DEST_DIR_DEBUG)/$(BASE_NAME).map ENV(EDK_TOOLS_PATH)/Bin/
Win32/ARM/RELEASE_RVCT31/$(INTRINSIC_FLAGS)


*_RVCT31_ARM_ASM_FLAGS       = $(ARCHASM_FLAGS) --apcs /interwork
*_RVCT31_ARM_PP_FLAGS        = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -E
*_RVCT31_ARM_VFRPP_FLAGS     = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -E  -DVFRCOMPILE -
-preinclude $(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h
*_RVCT31_ARM_MAKE_PATH       = nmake
*_RVCT31_ARM_SLINK_FLAGS     = --partial -o
  DEBUG_RVCT31_ARM_CC_FLAGS  = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) --c90 -c -g -O2 --
no_autoinline --asm --gnu --apcs /interwork --signed_chars --no_unaligned_access --
split_sections --enum_is_int --preinclude AutoGen.h --diag_warning 167
RELEASE_RVCT31_ARM_CC_FLAGS  = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) --c90 -c --
no_autoinline --asm --gnu --apcs /interwork --signed_chars --no_unaligned_access --
split_sections --enum_is_int --preinclude AutoGen.h --diag_warning 167

###################
# ARM definitions
###################
*_RVCT31_ARM_CC_PATH         = DEF(RVCT31_TOOLS_PATH)/armcc
*_RVCT31_ARM_SLINK_PATH      = DEF(RVCT31_TOOLS_PATH)/armlink
*_RVCT31_ARM_DLINK_PATH      = DEF(RVCT31_TOOLS_PATH)/armlink
*_RVCT31_ARM_ASM_PATH        = DEF(RVCT31_TOOLS_PATH)/armasm
*_RVCT31_ARM_PP_PATH         = DEF(RVCT31_TOOLS_PATH)/armcc
*_RVCT31_ARM_VFRPP_PATH      = DEF(RVCT31_TOOLS_PATH)/armcc
*_RVCT31_ARM_FROMELF_PATH    = DEF(RVCT31_TOOLS_PATH)/fromelf

############################################################################################
#
#
# ARM Tools & Cygwin
#
############################################################################################
#
#   ARMCYGWIN         - Tools from ARM in a Cygwin environment

*_RVCT31CYGWIN_*_*_FAMILY               = RVCT
*_RVCT31CYGWIN_*_*_BUILDRULEFAMILY      = RVCTCYGWIN

*_RVCT31CYGWIN_ARM_CCPATH_FLAG         = DEF(RVCT31CYGWIN_TOOLS_PATH)/armcc
*_RVCT31CYGWIN_ARM_SLINKPATH_FLAG      = DEF(RVCT31CYGWIN_TOOLS_PATH)/armlink
*_RVCT31CYGWIN_ARM_DLINKPATH_FLAG      = DEF(RVCT31CYGWIN_TOOLS_PATH)/armlink
*_RVCT31CYGWIN_ARM_ASMPATH_FLAG        = DEF(RVCT31CYGWIN_TOOLS_PATH)/armasm
*_RVCT31CYGWIN_ARM_PPPATH_FLAG         = DEF(RVCT31CYGWIN_TOOLS_PATH)/armcc
*_RVCT31CYGWIN_ARM_VFRPPPATH_FLAG      = DEF(RVCT31CYGWIN_TOOLS_PATH)/armcc
*_RVCT31CYGWIN_ARM_FROMELFPATH_FLAG    = DEF(RVCT31CYGWIN_TOOLS_PATH)/fromelf


#
# Use default values, or override in DSC file
#
*_RVCT31CYGWIN_ARM_ARCHCC_FLAGS    = --thumb --cpu ARM1176JZF-S
*_RVCT31CYGWIN_ARM_ARCHASM_FLAGS   = --cpu ARM1176JZF-S
*_RVCT31CYGWIN_ARM_ARCHDLINK_FLAGS =
```

```
*_RVCT31CYGWIN_ARM_PLATFORM_FLAGS  =
*_RVCT31CYGWIN_ARM_INTRINSIC_FLAGS = CompilerIntrinsicsLib.lib

  DEBUG_RVCT31CYGWIN_ARM_DLINK_FLAGS     = "$(DLINKPATH_FLAG)" $(ARCHDLINK_FLAGS) -
-entry $(IMAGE_ENTRY_POINT)  --ro-base 0 --no_scanlib --reloc --no_exceptions --
datacompressor off --strict --symbols --map --list `cygpath -m $(DEST_DIR_DEBUG)/
$(BASE_NAME).map` ENV(EDK_TOOLS_PATH)/Bin/CYGWIN_NT-5.1-i686/ARM/
DEBUG_RVCT31CYGWIN/$(INTRINSIC_FLAGS)
RELEASE_RVCT31CYGWIN_ARM_DLINK_FLAGS     = "$(DLINKPATH_FLAG)" $(ARCHDLINK_FLAGS) -
-entry $(IMAGE_ENTRY_POINT)  --ro-base 0 --no_scanlib --reloc --no_exceptions --
datacompressor off --strict --symbols --map --list `cygpath -m $(DEST_DIR_DEBUG)/
$(BASE_NAME).map` ENV(EDK_TOOLS_PATH)/Bin/CYGWIN_NT-5.1-i686/ARM/
DEBUG_RVCT31CYGWIN/$(INTRINSIC_FLAGS)

*_RVCT31CYGWIN_ARM_ASM_FLAGS       = "$(ASMPATH_FLAG)" $(ARCHASM_FLAGS) --apcs /
interwork
*_RVCT31CYGWIN_ARM_PP_FLAGS        = "$(CCPATH_FLAG)" $(ARCHCC_FLAGS)
$(PLATFORM_FLAGS) -E
*_RVCT31CYGWIN_ARM_VFRPP_FLAGS     = "$(CCPATH_FLAG)" $(ARCHCC_FLAGS)
$(PLATFORM_FLAGS) -E  -DVFRCOMPILE --preinclude `cygpath -m $(DEST_DIR_DEBUG)/
$(MODULE_NAME)StrDefs.h`
*_RVCT31CYGWIN_ARM_MAKE_PATH       = make
*_RVCT31CYGWIN_ARM_SLINK_FLAGS     = "$(SLINKPATH_FLAG)" --partial -o
  DEBUG_RVCT31CYGWIN_ARM_CC_FLAGS  = "$(CCPATH_FLAG)" $(ARCHCC_FLAGS)
$(PLATFORM_FLAGS) --c90 -c -g -O2 --no_autoinline --asm --gnu --apcs /interwork --
signed_chars --no_unaligned_access --split_sections --preinclude AutoGen.h --
diag_warning 167
RELEASE_RVCT31CYGWIN_ARM_CC_FLAGS  = "$(CCPATH_FLAG)" $(ARCHCC_FLAGS)
$(PLATFORM_FLAGS) --c90 -c --no_autoinline --asm --gnu --apcs /interwork --
```

```
signed_chars --no_unaligned_access --split_sections --preinclude AutoGen.h --
diag_warning 167

###################
# ARM definitions
###################
*_RVCT31CYGWIN_ARM_CC_PATH         = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_SLINK_PATH      = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_DLINK_PATH      = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_ASM_PATH        = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_PP_PATH         = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_VFRPP_PATH      = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py
*_RVCT31CYGWIN_ARM_FROMELF_PATH    = ENV(WORKSPACE)/BaseTools/Bin/CYGWIN_NT-5.1-
i686/armcc_wrapper.py

##############################################################################
#
#
# ARM GCC (GCC 4.1.1, etc.)
#
##############################################################################
#
#  ARMGCC         - ARM version of the GCC cross compiler

*_ARMGCC_*_*_FAMILY          = GCC
*_ARMGCC_*_MAKE_PATH         = make

###################
# ASL definitions
###################
*_ARMGCC_*_ASL_PATH                    = DEF(UNIX_IASL_BIN)
*_ARMGCCC_*_ASL_FLAGS                   =
*_ARMGCC_*_ASL_OUTFLAGS                = DEF(IASL_OUTFLAGS)
*_ARMGCC_*_ASLPP_FLAGS                 = -x c -E -P
*_ARMGCC_*_ASLCC_FLAGS                 = -x c
*_ARMGCC_*_ASLDLINK_FLAGS              = DEF(UNIXGCC_DLINK_FLAGS_COMMON) --entry
_ReferenceAcpiTable

###################
# ARM definitions
###################

*_ARMGCC_ARM_ASLCC_PATH      = DEF(ARMGCC_BIN)/arm-elf-gcc
*_ARMGCC_ARM_ASLDLINK_PATH   = DEF(ARMGCC_BIN)/arm-elf-ld
*_ARMGCC_ARM_ASLPP_PATH      = DEF(ARMGCC_BIN)/arm-elf-gcc

*_ARMGCC_ARM_CC_PATH         = DEF(ARMGCC_BIN)/arm-elf-gcc
*_ARMGCC_ARM_SLINK_PATH      = DEF(ARMGCC_BIN)/arm-elf-ar
*_ARMGCC_ARM_DLINK_PATH      = DEF(ARMGCC_BIN)/arm-elf-ld
```

```
*_ARMGCC_ARM_ASM_PATH        = DEF(ARMGCC_BIN)/arm-elf-as
*_ARMGCC_ARM_PP_PATH         = DEF(ARMGCC_BIN)/arm-elf-gcc
*_ARMGCC_ARM_VFRPP_PATH      = DEF(ARMGCC_BIN)/arm-elf-gcc

#
# Use default values, or override in DSC file
#
*_ARMGCC_ARM_ARCHCC_FLAGS    =
*_ARMGCC_ARM_ARCHASM_FLAGS   =
*_ARMGCC_ARM_ARCHDLINK_FLAGS =
*_ARMGCC_ARM_PLATFORM_FLAGS  =
*_ARMGCC_ARM_INTRINSIC_FLAGS = CompilerIntrinsicsLib.lib

  DEBUG_ARMGCC_ARM_ASM_FLAGS   = $(ARCHASM_FLAGS) -mlittle-endian -g
RELEASE_ARMGCC_ARM_ASM_FLAGS   = $(ARCHASM_FLAGS) -mlittle-endian

*_ARMGCC_ARM_PP_FLAGS    = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS)  -E -x assembler-with-
cpp -include $(DEST_DIR_DEBUG)/AutoGen.h
*_ARMGCC_ARM_VFRPP_FLAGS = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS)  -x c -E -P -
DVFRCOMPILE --include $(DEST_DIR_DEBUG)/$(MODULE_NAME)StrDefs.h

*_ARMGCC_ARM_MAKE_PATH   = make

*_ARMGCC_ARM_SLINK_FLAGS = -r
*_ARMGCC_ARM_DLINK_FLAGS =  $(ARCHDLINK_FLAGS) --emit-relocs --oformat=elf32-
littlearm -nostdlib -u $(IMAGE_ENTRY_POINT) -e $(IMAGE_ENTRY_POINT) -Map
$(DEST_DIR_DEBUG)/$(BASE_NAME).map
# --gc-sections

  DEBUG_ARMGCC_ARM_CC_FLAGS = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -mfpu=fpa -mlittle-
endian -g -O2 -mabi=aapcs -mapcs -fno-short-enums -save-temps -combine -fsigned-char
-fshort-wchar -fno-strict-aliasing -ffunction-sections -fdata-sections -Wall -
Werror -Wno-missing-braces -fomit-frame-pointer -c -include AutoGen.h
RELEASE_ARMGCC_ARM_CC_FLAGS = $(ARCHCC_FLAGS) $(PLATFORM_FLAGS) -mfpu=fpa -mlittle-
endian   -O2 -mabi=aapcs -mapcs -fno-short-enums -save-temps -combine -fsigned-char
```

```
-fshort-wchar -fno-strict-aliasing -ffunction-sections -fdata-sections -Wall -
Werror -Wno-missing-braces -fomit-frame-pointer -c -include AutoGen.h


###################
# VfrCompiler definitions
###################
*_*_*_VFR_PATH                     = VfrCompile
*_*_*_VFR_FLAGS                    = -l -n

###################
# OptionRom tool definitions
###################
*_*_*_OPTROM_PATH                  = EfiRom
*_*_*_OPTROM_FLAGS                 = -e

###################
# Asl Compiler definitions
###################
*_*_*_ASLCC_FLAGS                  = /GL- /Y- /TC /Dmain=ReferenceAcpiTable
*_*_*_ASLDLINK_FLAGS               = /NODEFAULTLIB /ENTRY:ReferenceAcpiTable /
SUBSYSTEM:CONSOLE
*_*_*_ASLPP_FLAGS                  = /nologo /EP /C
*_*_*_ASL_FLAGS                    =

###################
# LzmaCompress tool definitions
###################
*_*_*_LZMA_PATH          = LzmaCompress
*_*_*_LZMA_GUID          = EE4E5898-3914-4259-9D6E-DC7BD79403CF

###################
# TianoCompress tool definitions
###################
*_*_*_TIANO_PATH         = TianoCompress
*_*_*_TIANO_GUID         = A31280AD-481E-41B6-95E8-127F4C984779
```

Appendix C
target.txt

The following is the default version of the target.txt file. Note that the lines wrap in this document, however no line wrapping is permitted in the target.txt file.

```
#
#  Copyright (c) 2006-2007, Intel Corporation
#
#  All rights reserved. This program and the accompanying materials
#  are licensed and made available under the terms and conditions of the BSD
#  License which accompanies this distribution.  The full text of the license
#  may be found at
#  http://opensource.org/licenses/bsd-license.php

#  THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
#  WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.
#
#  Filename: target.template
#
#  ALL Paths are Relative to WORKSPACE

#  Separate multiple LIST entries with a SINGLE SPACE character, do not use comma
#  characters.
#  Un-set an option by either commenting out the line, or not setting a value.

#
#  PROPERTY              Type       Use         Description
#  ----------------      --------   --------    --------------------------------
#  ACTIVE_PLATFORM       Filename   Recommended Specify the WORKSPACE relative
#                                               Path and Filename of the platform
#                                               DSC file that will be used for
#                                               the build
#                                               This line is required if and only
#                                               if the current working directory
#                                               does not contain one or more DSC
#                                               files.
ACTIVE_PLATFORM       = Nt32Pkg/Nt32Pkg.dsc

#  TARGET                List       Optional    Zero or more of the following:
#                                               DEBUG, RELEASE, a User Defined
#                                               word, separated by a space
#                                               character.
#                                               If the line is missing or no
#                                               value is specified, all valid
#                                               targets specified in the DSC file
#                                               will attempt to be built.
#                                               The following line will build
#                                               all platform DEBUG targets.
TARGET                = DEBUG


#  TARGET_ARCH           List       Optional    What kind of architecture is the
#                                               binary being target for.
#                                               One, or more, of the following,
#                                               IA32, IPF, X64, EBC or ARM.
#                                               Multiple values can be specified
#                                               on a single line, using space
#                                               characters to separate the
#                                               values.  These are used during
```

```
#                                          the parsing of an DSC file,
#                                          restricting the build output
#                                          target(s.)
#                                          The Build Target ARCH is
#                                          determined by a logical AND of:
#                               target.txt: TARGET_ARCH values
#                                 DSC file: [Defines] SUPPORTED_ARCHITECTURES tag
#                             Command-line: -a ARCH option
#                                          If not specified, then all valid
#                                          architectures specified in the
#                                          DSC file, for which tools are
#                                          available, will be built.
TARGET_ARCH           = IA32

#  TOOL_DEFINITION_FILE  Filename  Optional   Specify the name of the filename to
#                                            use for specifying the tools to use
#                                            for the build.  If not specified,
#                                            the file:
#                                            WORKSPACE/Conf/tools_def.txt will
#                                            be used for the build.  This file
#                                            MUST be located in the
#                                            WORKSPACE/Conf directory.
TOOL_CHAIN_CONF       = Conf/tools_def.txt

#  TOOL_CHAIN_TAG        List      Optional   Specify the name(s) of the
#                                            tools_def.txt TagName to use.
#                                            If not specified, all applicable
#                                            Tool Chain Tag name tools will be
#                                            used for the build.  The list uses
#                                            space character separation.
TOOL_CHAIN_TAG        = MYTOOLS

#  MULTIPLE_THREAD       FLAG      Optional   Flag to enable multi-thread build.
#                                            If not specified, default is
#                                            "Disable". If your computer is
#                                            multi-core or has multiple
#                                            CPUs, enabling this feature will
#                                            bring a performance benefit.
#                                            This feature is only available for
#                                            "spawn" build mode, and only for
#                                            PLATFORM builds. The clean,
#                                            cleanall or stand-alone module
#                                            build is still single threaded.
MULTIPLE_THREAD       = Enable

# MAX_CONCURRENT_THREAD_NUMBER  NUMBER  Optional  The number of concurrent
#                                                 threads. Default is 2.
#                                                 Recommend setting this value to
#                                                 one more than the number of your
#                                                 computer cores pr CPUs.
MAX_CONCURRENT_THREAD_NUMBER = 2

# BUILD_RULE_CONF      Filename  Optional   Specify the name of the filename to
#                                           use for the build rules that are
#                                           followed when generating Makefiles.
```

```
#                                        If not specified, the file:
#                                       WORKSPACE/Conf/build_rule.txt will be
#                                        used
BUILD_RULE_CONF = Conf/build_rule.txt

# ADD_BUILD_RULE       Filename  Optional   Specify the name of the filename to
#                                           use for additional build rules that
#                                           are followed when when generating
#                                           Makefiles.
#                                           The build tools will "append" this
#                                           file to the end of the
#                                           BUILD_RULE_CONF file.
#ADD_BUILD_RULE
```

Appendix D
build_rule.txt

The following is the default version of the build_rule.txt file. Note that the lines wrap in this document, however no line wrapping is permitted in the build_rule.txt file.

```
#
#  Copyright (c) 2007-2009, Intel Corporation
#  Portions copyright (c) 2008 - 2009, Apple Inc. All rights reserved.
#  All rights reserved. This program and the accompanying materials
#  are licensed and made available under the terms and conditions of the BSD License
#  which accompanies this distribution.  The full text of the license may be found at
#  http://opensource.org/licenses/bsd-license.php

#  THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
#  WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.
#
#  Filename: build_rule.template
#

## Syntax
#
# "*" is used to indicate that the source files will be processed at the same time.
# "?" is used to indicate that the source files will be processed one by one.
#
# "[" <File-Type>[.<Build-Type>][.<Arch>][, <File-Type>[.<Build-Type>][.<Arch>]] "]" <EOL>
#   <InputFile[.<ToolChainFamily>]> <EOL>
#       (?|*).<File-Extension> [(\n|,) (?|*).<File-Extension>]
#
#   <OutputFile[.<ToolChainFamily>]> <EOL>
#       <FileFullPath>
#
#   <ExtraDependency>
#       <FileFullPath>
#
#   <Command[.<ToolChainFamily>]> <EOL>
#       <Command1>
#       [<Command2>]
#
# <Build-Type> is the MODULE_TYPE in EDK2 or COMPONENT_TYPE  in EDK.
# Missing <InputFile> will cause an exception and break build.
# Missing <Command> will cause that related build target won't be generated but
#   won't break build.
#

## Placeholders for string substitution
#
#   ${src}      Source file(s) to be built (full path)
#   ${s_path}   Source file directory (absolute path)
#   ${s_dir}    Source file relative directory within a module
#               (Note: ${s_dir} is always equals to "." if source file is given in absolute
path.)
#   ${s_name}   Source file name without path
#   ${s_base}   Source file name without extension and path
#   ${s_ext}    Source file extension
#
#   ${dst}      Destination file(s) built from ${src} (full path)
#   ${d_path}   Destination file directory (absolute path)
#   ${d_name}   Destination file name without path
#   ${d_base}   Destination file name without extension and path
#   ${d_ext}    Destination file extension
#
#   (+)         Directory separator
#

## Macro
#   $(WORKSPACE)        Workspace directory
```

```
#   $(OUTPUT_DIR)       Directory for intermediate files for building a module
#   $(DEBUG_DIR)        Directory for files used to debug a module
#   $(BUILD_DIR)        All files for building a platform will be put in this directory
#   $(BIN_DIR)          Common directory for executable files
#   $(FV_DIR)           Directory to store flash image files
#   $(INC)              Search path of current module
#   $(INC_LIST)         A file containing search pathes of current module
#   $(LIBS)             Static library files of current module
#   $(<tool>_FLAGS)     Tools flags of current module
#   $(MODULE_NAME)      Current module name
#   $(MODULE_TYPE)      Current module type
#   $(MODULE_GUID)      Current module guid
#   $(ARCH)             Architecture of current module
#   $(TOOLCHAIN)        Toolchain used to build current module
#   $(TARGET)           Target of current module (DEBUG/RELEASE)
#   $(<tool>)           Path of tool
#   $(EDK_TOOLS_PATH)   Path of build tools
#   $(<FILE_TYPES>)     File list of each file type
#                       (Note: The macro name is derived from file type name. For example,
#                        C-Code-File will have C_CODE_FILES macro.)
#   $(<FILE_TYPES_LIST>) Macro point to a file containing list of files of a file type
#                       (
#                        Note: The macro and file name are derived from file type name.
#                        For example, C-Code-File will have C_CODE_FILES_LIST macro pointing
#                        to $(OUTPUT_DIR)/c_code_files.lst. The list file and macro name
#                        will be generated only when this macro is used in command line.
#                        This is intended to get over the long command line limitation.
#                       )
#
#   $(CP)      copy command
#   $(MV)      move command
#   $(RM)      delete command
#   $(MD)      create dir command
#   $(RD)      remove dir command
#

## Reserved File-Type
#
# Don't change following names of file types and their associated files,
# which are also used in tools' code
#
#   C-Code-File
#   C-Header-File
#   Dynamic-Library-File
#   Static-Library-File
#   Visual-Form-Representation-File
#   Unicode-Text-File
#

[C-Code-File]
    <InputFile>
        ?.c
        ?.C
        ?.cc
        ?.CC
        ?.cpp
        ?.Cpp
        ?.CPP

    <ExtraDependency>
        $(MAKE_FILE)
```

```
    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj

    <Command.MSFT, Command.INTEL>
        "$(CC)" /Fo${dst} $(CC_FLAGS) $(INC) ${src}

    <Command.GCC, Command.RVCT>
        # For RVCTCYGWIN CC_FLAGS must be first to work around pathing issues
        "$(CC)" $(CC_FLAGS) -o ${dst} $(INC) ${src}
        "$(SYMRENAME)" $(SYMRENAME_FLAGS) ${dst}

[C-Header-File]
    <InputFile>
        *.h, *.H

    <OutputFile>

    <Command>


[Assembly-Code-File.COMMON.COMMON]
    <InputFile.MSFT, InputFile.INTEL, InputFile.RVCT>
        ?.asm, ?.Asm, ?.ASM

    <InputFile.GCC>
        ?.S, ?.s

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj

    <Command.MSFT, Command.INTEL>
        "$(PP)" $(PP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
        Trim --source-code --convert-hex -o ${d_path}(+)${s_base}.iii ${d_path}(+)${s_base}.i
        "$(ASM)" /Fo${dst} $(ASM_FLAGS) /I${s_path} $(INC) ${d_path}(+)${s_base}.iii

    <Command.GCC, Command.RVCT>
        "$(PP)" $(PP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
        Trim --source-code -o ${d_path}(+)${s_base}.iii ${d_path}(+)${s_base}.i
        # For RVCTCYGWIN ASM_FLAGS must be first to work around pathing issues
        "$(ASM)" $(ASM_FLAGS) -o ${dst} $(INC) ${d_path}(+)${s_base}.iii

[Assembly-Code-File.COMMON.IPF]
    <InputFile>
        ?.s

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj

    <Command.MSFT, Command.INTEL>
        "$(PP)" $(PP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
        Trim --source-code -o ${d_path}(+)${s_base}.iii ${d_path}(+)${s_base}.i
        # For RVCTCYGWIN ASM_FLAGS must be first to work around pathing issues
        "$(ASM)" $(ASM_FLAGS) -o ${dst} ${d_path}(+)${s_base}.iii
```

```
[Visual-Form-Representation-File]
    <InputFile>
        ?.vfr
        ?.Vfr
        ?.VFR

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(DEBUG_DIR)(+)${s_dir}(+)${s_base}.c

    <Command>
        -$(MD) $(OUTPUT_DIR)(+)${s_dir} > NUL 2>&1
        "$(VFRPP)" $(VFRPP_FLAGS) $(INC) ${src} > $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        -$(MD) ${d_path} > NUL 2>&1
        "$(VFR)" $(VFR_FLAGS) --output-directory ${d_path}
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i

[Object-File]
    <InputFile>
        *.obj
        *.o

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME).lib

    <Command.MSFT, Command.INTEL>
        "$(SLINK)" $(SLINK_FLAGS) /OUT:${dst} @$(OBJECT_FILES_LIST)

    <Command.GCC>
        "$(SLINK)" -cr ${dst} $(SLINK_FLAGS) @$(OBJECT_FILES_LIST)

    <Command.RVCT>
        "$(SLINK)" $(SLINK_FLAGS) ${dst} --via $(OBJECT_FILES_LIST)

    <Command.RVCTCYGWIN>
        # $(OBJECT_FILES_LIST) has wrong paths for cygwin
        "$(SLINK)" $(SLINK_FLAGS) ${dst} $(OBJECT_FILES)

    <Command.XCODE>
        "$(SLINK)" $(SLINK_FLAGS) ${dst} -filelist $(OBJECT_FILES_LIST)

[Static-Library-File]
    <InputFile>
        *.lib

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(DEBUG_DIR)(+)$(MODULE_NAME).dll

    <Command.MSFT, Command.INTEL>
        "$(DLINK)" /OUT:${dst} $(DLINK_FLAGS) $(DLINK_SPATH) @$(STATIC_LIBRARY_FILES_LIST)

    <Command.GCC>
        "$(DLINK)" -o ${dst} $(DLINK_FLAGS) -\( $(DLINK_SPATH) @$(STATIC_LIBRARY_FILES_LIST) -
\) $(DLINK2_FLAGS)
        "$(OBJCOPY)" $(OBJCOPY_FLAGS) ${dst}
```

```
    <Command.RVCT>
        "$(DLINK)" $(DLINK_FLAGS) -o ${dst} $(DLINK_SPATH) --via $(STATIC_LIBRARY_FILES_LIST)
$(DLINK2_FLAGS)

    <Command.RVCTCYGWIN>
        #$(STATIC_LIBRARY_FILES_LIST) has wrong paths for cygwin
        "$(DLINK)" $(DLINK_FLAGS) -o ${dst} $(DLINK_SPATH) $(STATIC_LIBRARY_FILES)
$(DLINK2_FLAGS)

    <Command.XCODE>
        "$(DLINK)" $(DLINK_FLAGS) -o ${dst} $(DLINK_SPATH) -filelist
$(STATIC_LIBRARY_FILES_LIST)  $(DLINK2_FLAGS)


[Static-Library-File.USER_DEFINED]
    <InputFile>
        *.lib

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(DEBUG_DIR)(+)$(MODULE_NAME)

    <Command.MSFT, Command.INTEL>
        "$(DLINK)" $(DLINK_FLAGS) $(DLINK_SPATH) @$(STATIC_LIBRARY_FILES_LIST)

    <Command.GCC>
        "$(DLINK)" $(DLINK_FLAGS) -\( $(DLINK_SPATH) @$(STATIC_LIBRARY_FILES_LIST) -\)
$(DLINK2_FLAGS)

    <Command.RVCT>
        "$(DLINK)" $(DLINK_FLAGS) -o ${dst} $(DLINK_SPATH) --via $(STATIC_LIBRARY_FILES_LIST)
$(DLINK2_FLAGS)

    <Command.RVCTCYGWIN>
        #$(STATIC_LIBRARY_FILES_LIST) has the wrong paths for cygwin
        "$(DLINK)" $(DLINK_FLAGS) -o ${dst} $(DLINK_SPATH) $(STATIC_LIBRARY_FILES)
$(DLINK2_FLAGS)

    <Command.XCODE>
        "$(DLINK)" -o ${dst} $(DLINK_FLAGS)  $(DLINK_SPATH) -filelist
$(STATIC_LIBRARY_FILES_LIST)  $(DLINK2_FLAGS)


[Dynamic-Library-File]
    <InputFile>
        ?.dll

    <OutputFile>
        $(DEBUG_DIR)(+)$(MODULE_NAME).efi

    <Command.MSFT, Command.INTEL, Command.GCC, Command.RVCT>
        GenFw -e $(MODULE_TYPE) -o ${dst} ${src}
        $(CP) ${dst} $(OUTPUT_DIR)
        $(CP) ${dst} $(BIN_DIR)
        -$(CP) $(DEBUG_DIR)(+)*.map $(OUTPUT_DIR)

    <Command.XCODE>
        # tool to convert Mach-O to PE/COFF
        "$(MTOC)" -subsystem $(MODULE_TYPE)  $(MTOC_FLAGS)  ${src}
```

```
$(DEBUG_DIR)(+)$(MODULE_NAME).pecoff
        # create symbol file for GDB debug
        -$(DSYMUTIL) ${src}
        GenFw -e $(MODULE_TYPE) -o ${dst} $(DEBUG_DIR)(+)$(MODULE_NAME).pecoff
        $(CP) ${dst} $(OUTPUT_DIR)
        $(CP) ${dst} $(BIN_DIR)
        -$(CP) $(DEBUG_DIR)(+)*.map $(OUTPUT_DIR)

[Dependency-Expression-File]
    <InputFile>
        ?.dxs, ?.Dxs, ?.DXS

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME).depex

    <Command>
        "$(PP)" $(CC_FLAGS) $(APP_FLAGS) $(INC) ${src} > $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        Trim --source-code -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        GenDepex -t $(MODULE_TYPE) -o ${dst} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii

[Acpi-Source-Language-File]
    <InputFile>
        ?.asl, ?.Asl, ?.ASL

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.aml

    <ExtraDependency>
        $(MAKE_FILE)

    <Command.MSFT, Command.INTEL>
        Trim --asl-file -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i ${src}
        "$(ASLPP)" $(ASLPP_FLAGS) $(INC) /I${s_path} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i >
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii
        "$(ASL)" $(ASL_FLAGS) $(ASL_OUTFLAGS)${dst} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii

    <Command.GCC>
        Trim --asl-file -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i ${src}
        "$(ASLPP)" $(ASLPP_FLAGS) $(INC) -I${s_path} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i >
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii
        "$(ASL)" $(ASL_FLAGS) $(ASL_OUTFLAGS)${dst} $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.iii

[C-Code-File.AcpiTable]
    <InputFile>
        ?.c

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.acpi

    <ExtraDependency>
        $(MAKE_FILE)

    <Command.MSFT, Command.INTEL>
        "$(ASLCC)" /Fo$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(CC_FLAGS) $(ASLCC_FLAGS)
$(INC) ${src}
        "$(ASLDLINK)" /OUT:$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(ASLDLINK_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
        GenFw -o ${dst} -c $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll

    <Command.GCC>
```

```
        "$(ASLCC)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(CC_FLAGS) $(ASLCC_FLAGS)
$(INC) ${src}
        "$(ASLDLINK)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(ASLDLINK_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
        GenFw -o ${dst} -c $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll

[Acpi-Table-Code-File]
    <InputFile>
        ?.aslc, ?.act

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.acpi

    <ExtraDependency>
        $(MAKE_FILE)

    <Command.MSFT, Command.INTEL>
        "$(ASLCC)" /Fo$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(CC_FLAGS) $(ASLCC_FLAGS)
$(INC) ${src}
        "$(ASLDLINK)" /OUT:$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(ASLDLINK_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
        GenFw -o ${dst} -c $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll

    <Command.GCC>
        "$(ASLCC)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(CC_FLAGS) $(ASLCC_FLAGS)
$(INC) ${src}
        "$(ASLDLINK)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(ASLDLINK_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
        GenFw -o ${dst} -c $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll

    <Command.XCODE>
        "$(ASLCC)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj  $(ASLCC_FLAGS) $(INC) ${src}
        "$(ASLDLINK)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(ASLDLINK_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
        "$(MTOC)" -subsystem $(MODULE_TYPE)  $(MTOC_FLAGS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.dll $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.efi
        GenFw -o ${dst} -c $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.efi


[Masm16-Code-File]
    <InputFile>
        ?.asm16, ?.Asm16, ?.ASM16

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_base}.com

    <Command.MSFT, Command.INTEL>
        cd $(OUTPUT_DIR)(+)${s_dir}
        "$(ASM)" /nologo /c /omf /Fo$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj ${src}
        "$(ASMLINK)" $(ASMLINK_FLAGS) $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj,${dst},,,,

    <Command.GCC>
      "$(PP)" $(PP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
      Trim --source-code -o ${d_path}(+)${s_base}.iii ${d_path}(+)${s_base}.i
      "$(ASM)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(ASM_FLAGS) $(INC)
${d_path}(+)${s_base}.iii
      "$(DLINK)" -o ${dst} $(DLINK_FLAGS) -( $(DLINK_SPATH) $(LIBS)
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj -)
```

```
    <Command.XCODE>
      "$(PP)" $(PP_FLAGS) $(INC) ${src} > ${d_path}(+)${s_base}.i
      Trim --source-code -o ${d_path}(+)${s_base}.iii ${d_path}(+)${s_base}.i
      "$(ASM)" -o $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj $(ASM_FLAGS) $(INC)
${d_path}(+)${s_base}.iii
      "$(SLINK)" $(SLINK_FLAGS) $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.slib
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.obj
      otool -t $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.slib | hex2bin.py ${dst}


[Microcode-File]
    <InputFile>
        ?.txt, ?.TXT, ?.Txt, ?.mut

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_base}.mcb

    <Command>
        GenFw -o ${dst} -m ${src}

[Microcode-Binary-File]
    <InputFile>
        *.mcb

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME).bin

    <Command>
        GenFw -o ${dst} -j $(MICROCODE_BINARY_FILES)
        -$(CP) ${dst} $(BIN_DIR)

[EFI-Image-File]
    <InputFile>
        ?.efi, ?.Efi, ?.EFI

    <OutputFile>

    <Command>

[Unicode-Text-File]
    <InputFile>
        *.uni, *.Uni, *.UNI

    <OutputFile>
        $(DEBUG_DIR)(+)AutoGen.c
        $(DEBUG_DIR)(+)$(MODULE_NAME)StrDefs.h

    <Command>

[Efi-Image.UEFI_OPTIONROM]
    <InputFile>
        ?.efi, ?.EFI, ?.Efi

    <OutputFile>
        $(BIN_DIR)(+)$(MODULE_NAME).rom

    <Command>
        $(OPTROM) -i $(PCI_DEVICE_ID) -f $(PCI_VENDOR_ID) -l $(PCI_CLASS_CODE) -r
$(PCI_REVISION) -o $dst $(OPTROM_FLAGS) $src
```

```
[Unicode-Text-File.UEFI_HII]
    <InputFile>
        *.uni, *.Uni, *.UNI

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME)StrDefs.hpk
        $(DEBUG_DIR)(+)$(MODULE_NAME)StrDefs.h

    <Command>

[Visual-Form-Representation-File.UEFI_HII]
    <InputFile>
        ?.vfr
        ?.Vfr
        ?.VFR

    <ExtraDependency>
        $(MAKE_FILE)

    <OutputFile>
        $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.hpk

    <Command>
        -$(MD) $(OUTPUT_DIR)(+)${s_dir} > NUL 2>&1
        "$(VFRPP)" $(VFRPP_FLAGS) $(INC) ${src} > $(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i
        -$(MD) $(OUTPUT_DIR)(+)${s_dir} > NUL 2>&1
       "$(VFR)" $(VFR_FLAGS) --create-ifr-package --output-directory $(OUTPUT_DIR)(+)${s_dir}
$(OUTPUT_DIR)(+)${s_dir}(+)${s_base}.i

[Hii-Binary-Package.UEFI_HII]
    <InputFile>
        *.hpk

    <OutputFile>
        $(OUTPUT_DIR)(+)$(MODULE_NAME)hii.lib

    <Command.MSFT, Command.INTEL>
        GenFw -o $(OUTPUT_DIR)(+)$(MODULE_NAME)hii.rc -g $(MODULE_GUID) --hiipackage
$(HII_BINARY_PACKAGES)
        "$(RC)" /Fo${dst} $(OUTPUT_DIR)(+)$(MODULE_NAME)hii.rc

    <Command.GCC>
        GenFw -o $(OUTPUT_DIR)(+)$(MODULE_NAME)hii.rc -g $(MODULE_GUID) --hiipackage
$(HII_BINARY_PACKAGES)
        "$(RC)" (RC_FLAGS) $(OUTPUT_DIR)(+)$(MODULE_NAME)hii.rc ${dst}
```

Appendix E
build.exe command

This section describes the build.exe command line tool's options. Build.exe is generated from Python code.

E.1 Overview

Under normal circumstances, the build tool will:

1. Process the command-line options
2. Parse the meta-data files
3. Generate the C files
4. Generate the Makefiles
5. Call the make command.

E.2 Makefile actions

The Makefile is responsible for compilation of the source code and executing the genFw command on the intermediate object files in order to create the .efi files.

The actions taken by the Makefile are:

1. Create the build output directories
2. Build the libraries
3. Build the modules
4. Call the genfds tool
5. The last step of the makefile processing is to call the GenFds tool that will:
   a Process the command-line options
   b Parse the meta-data files
   c Create FFS, Capsules, FV images and the final FD image(s).

E.3 Build Targets and options

In order to provide flexibility, the build command supports stopping the build process after specific actions have taken place. These targets will ensure that all previously required actions have been completed. New for this release is the implemenation of targets that permit processing files for only one given step, such that previous steps are NOT processed. Table 28 provides the descriptions of targets supported by the build, as well as the GenFds tools.

Note: *The flag, --skip-autogen, is required to prevent the build tool from re-creating the autogenerated C and makefiles.*

Table 28. Build Targets and Comand-line Options

Target	Description	Target	Description
genc	Generates the C code files (autogen.c, autogen.h and module.depex) then stops		
genmake	Generates the C code files (autogen.c, autogen.h and module.depex) then the Makefiles, then stops	makefiles	Generates only the Makefiles (assumes that genc has completed and all C code files exist).
libraries	Generates the C code files, the Makefiles, then generates the object files for libraries	genlibs and --skip-autogen	Will only generate the object files for libraries
modules	Generates the C code files, the Makefiles, generates the object files for libraries, generates the object files for the modules, then links them, then calls genfw for each of the intermediate final objects to create .efi files	genmodules and --skip-autogen	Generates the object files for the modules, then links them with library object files that must already exist into intermediate object files
		genefi and --skip-autogen	Processes the intermediate PE32/PE32+ object files that must already exist, modifying the header to match **EFI_IMAGE_FILE_HEADER** with output files having a .efi extension.
fds	Generates the C code files, the Makefiles, generates the object files for libraries, generates the object files for the modules, links them, calls genfw for each of the intermediate final objects to create .efi files, generates SECTION files, generates FFS files, generates FV files and finally generates FD files	gensections and --skip-autogen	For each .efi file, will call the genSec
		genffs and --skip-autogen	For each module, assemble the section files into an FFS file.
		genfvs and --skip-autogen	For each FV listed in the FDF file, creat the FV file.
		genfd and --skip-autogen	Assembles the existing FV files and other content into the ROM (FD) image.
		gencapsule and --skip-autogen	Generates a Capsule Image
		genOptionRom and --skip-autogen	Generates an OptionRom

E.4 Usage

```
Usage: build.exe [options] [target]
```

```
Copyright (c) 2007, Intel Corporation  All rights reserved.

Options:
  --version
                        show program's version number and exit
  -h, --help
                        show this help message and exit
  -a TARGETARCH, --arch=TARGETARCH
                        ARCHS is one of list: IA32, X64, IPF or EBC, which
                        overrides target.txt's TARGET_ARCH definition. To
                        specify more archs, please repeat this option.
  -p PLATFORMFILE, --platform=PLATFORMFILE
                        Build the platform specified by the DSC file name
                        argument, overrides target.txt's ACTIVE_PLATFORM
                        definition.
  -m MODULEFILE, --module=MODULEFILE
                        Build the module specified by the INF file name
                        argument.
  -b BUILDTARGET, --buildtarget=BUILDTARGET
                        BuildTarget is one of list: DEBUG, RELEASE, which
                        overrides target.txt's TARGET definition. To specify
                        more TARGET, please repeat this option.
  -t TOOLCHAIN, --tagname=TOOLCHAIN
                        Using the Tool Chain Tagname to build the platform,
                        overrides target.txt's TOOL_CHAIN_TAG definition.
  -s, --spawn
                        If this flag is specified, as soon as a module can be
                        built, the build will start, without waiting for
                        AutoGen to complete remaining modules. While this
                        option provides feedback that looks fast, due to
                        overhead of the AutoGen function, this option is
                        slower than letting AutoGen complete before starting
                        the MAKE phase.
  -n THREADNUMBER
                        Build the platform using multi-threaded compiler, this
                        option must combine with spawn option. The value
                        overrides target.txt's MULTIPLE_THREAD and
                        MAX_CONCURRENT_THREAD_NUMBER, less than 2 will disable
                        multi-thread builds.
  -f FDFFILE, --fdf=FDFFILE
                        The name of the FDF file to use, which overrides the
                        setting in the DSC file.
  -r ROMIMAGE, --rom-image=ROMIMAGE
                        The name of FD to be generated. The name must be from
                        [FD] section in FDF file.
  -i FVIMAGE, --fv-image=FVIMAGE
                        The name of FV to be generated. The name must be from
                        [FV] section in FDF file.
  -k, --msft
                        Make Option: Generate only NMAKE Makefiles: Makefile
  -g, --gcc
                        Make Option: Generate only GMAKE Makefiles: GNUmakefile
  -l, --all
                        Make Option: Generate both NMAKE and GMAKE makefiles.
  -w, --warning-as-error
```

```
                    Treat warning in tools as error.
-j LOGFILE, --log=LOGFILE
                    Putlog in specified file as well as on console.
-q, --quiet
                    Disable all messages except FATAL ERRORS.
-v, --verbose
                    Turn on verbose output with informational messages
                    printed, including library instances selected, final
                    dependency expression, and warning messages, etc.
-d DEBUG, --debug=DEBUG
                      Enable debug messages at specified level.
```

Appendix F
NT32 Platform Emulation Envoronment

The NT32Pkg provides a platform emulation environment that executes on windows platform. The reference build program is used to start the emulation environment after it has been built.

Prior to building the platform: Nt32Pkg\Nt32Pkg.dsc, the user may want to modify PCD settings in the file. The following PCDs control the mappings of your system environment to the emulation environment.

```
PcdWinNtSerialPort|L"COM1!COM2"|VOID*|18
```

This maps the serial port to COM1 or COM2 (if COM1 is not available).

```
PcdWinNtFileSystem | \
  L".!..\\..\\..\\..\\EdkShellBinPkg\\bin\\ia32\\Apps"|VOID*|106
```

This shows the location of the shell applications.

```
PcdWinNtGop|L"UGA Window 1!UGA Window 2"|VOID*|50
```

This defines label for the two windows that are started.

```
PcdWinNtConsole|L"Bus Driver Console Window"|VOID*|50
```

This defines label for the windows that are started.

```
PcdWinNtVirtualDisk|L"FW;40960;512"|VOID*|24
```

This defines the max and block sizes for the virtual disk drive that is created.

```
PcdWinNtMemorySize|L"64!64"|VOID*|10
```

This defines the memory available for the emulator in MBytes.

```
PcdWinNtPhysicalDisk | \
  L"a:RW;2880;512!d:RO;307200;2048!j:RW;262144;512"|VOID*|100
```

This defines the available storage devices that should be present at startup, A:, D: and J: - you may want to change the drive letters to match the development environment - note that you should not use the C: drive, as you could inadvertently wipe it out.

```
PcdWinNtUga|L"UGA Window 1!UGA Window 2"|VOID*|50
```

This defines label for the two windows that are started

Appendix G
Firmware Volume INF

The Firmware Volume INF file is generated by the EDK II build tools as an intermediate file between the code generation stage and the final image creation stage.

G.1 Firmware Volume INF Description

The Firmware Volume INF files are generated by tool based on content from Platform description files (DSC) and Flash definition files (FDF) and may contain these three sections: `[options]`, `[attributes]` and `[files]`.

This file is an input to the GenFvImage utility.

```
<FIRMWARE_VOLUME_INF>  ::=  [<options>]
                            [<attributes>]
                            [<files>]
```

G.2 [Attributes] Section

Summary

Defines the `[Attributes]` tag is found only in Firmware Volume INF files. This file is created by the tools and is an input to the GenFvImage utility. Refer to the document, "*Intel® Platform Innovation Framework for EFI, Firmware Volume Block Specification*" for more information on these values. This is an optional section.

Prototype

```
<attributes>     ::=  "[attributes]" <EOL>
                      <expression>

<expression>     ::=  ["EFI_READ_DISABLED_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_READ_ENABLED_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_READ_STATUS" "=" <TrueFalse> <EOL>]
                      ["EFI_WRITE_DISABLED_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_WRITE_ENABLED_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_WRITE_STATUS" "=" <TrueFalse> <EOL>]
                      ["EFI_LOCK_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_LOCK_STATUS" "=" <TrueFalse> <EOL>]
                      ["EFI_ERASE_POLARITY" "=" <ZeroOne> <EOL>]
                      ["EFI_STICK_WRITE" "=" <TrueFalse> <EOL>]
                      ["EFI_MEMORY_MAPPED" "=" <TrueFalse> <EOL>]
                      ["EFI_ALIGNMENT_CAP" "=" <TrueFalse> <EOL>]
                      ["EFI_ALIGNMENT_2" "=" <TrueFalse> <EOL>]
                      ["EFI_ALIGNMENT_4" "=" <TrueFalse> <EOL>]
                      ["EFI_ALIGNMENT_8" "=" <TrueFalse> <EOL>]
                      ["EFI_ALIGNMENT_16" "=" <TrueFalse> <EOL>]
```

```
                        ["EFI_ALIGNMENT_32" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_64" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_128" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_256" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_512" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_1K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_2K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_4K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_8K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_16K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_32K" "=" <TrueFalse> <EOL>]
                        ["EFI_ALIGNMENT_64K" "=" <TrueFalse> <EOL>]

<TrueFalse>      ::=  {<ZeroOne>} {<TF>}

<TF>             ::=  {<True>} {<False>}

<True>           ::=  {"TRUE"} {"True"} {"true"}

<False>          ::=  {"FALSE"} {"False"} {"false"}

<ZeroOne>        ::=  {"0"} {"1"}

<EOL>            ::=  end of line
```

Example

```
[attributes]
EFI_READ_DISABLED_CAP    = TRUE
EFI_READ_ENABLED_CAP     = TRUE
EFI_READ_STATUS          = TRUE
EFI_WRITE_DISABLED_CAP   = TRUE
EFI_WRITE_ENABLED_CAP    = TRUE
EFI_WRITE_STATUS         = TRUE
EFI_LOCK_CAP             = TRUE
EFI_LOCK_STATUS          = FALSE
EFI_STICKY_WRITE         = TRUE
EFI_MEMORY_MAPPED        = TRUE
EFI_ERASE_POLARITY       = 1
EFI_ALIGNMENT_CAP        = TRUE
EFI_ALIGNMENT_2          = TRUE
EFI_ALIGNMENT_4          = TRUE
EFI_ALIGNMENT_8          = TRUE
EFI_ALIGNMENT_16         = TRUE
EFI_ALIGNMENT_32         = TRUE
EFI_ALIGNMENT_64         = TRUE
EFI_ALIGNMENT_128        = TRUE
EFI_ALIGNMENT_256        = TRUE
EFI_ALIGNMENT_512        = TRUE
EFI_ALIGNMENT_1K         = TRUE
EFI_ALIGNMENT_2K         = TRUE
EFI_ALIGNMENT_4K         = TRUE
EFI_ALIGNMENT_8K         = TRUE
EFI_ALIGNMENT_16K        = TRUE
EFI_ALIGNMENT_32K        = TRUE
EFI_ALIGNMENT_64K        = TRUE
```

G.3 [Files] Section

Summary

Defines the `[files]` tag is found only in Firmware Volume INF files. This file is created by the build utility and is an input to the GenFvImage utility.

Prototype

```
<files>           ::=  "[files]" <EOL>
                       <expression>+

<expression>      ::=  <Filename> [<COMPONENT_TYPE>] [<FVS>]
                         [<FFSEXT>] ["PROCESSOR=" <arch>] [<APRORI>]
                         [<EFN>] <EOL>

<Filename>        ::=  <PATH> <Word> <Extension>

<COMPONENT_TYPE>  ::=  Refer to Table "Component (module) Types"

<PATH>            ::=  [[".."]{0,1} "\"]* {<Word> {"\"}{0,1}}*
```

```
<arch>            ::=  {IA32} {X64} {IPF} {EBC}

<FVS>             ::=  "FVs=" <FvImageName>[", <FvImageName>]*

<FvImageName>     ::=  <Word>

<FFSEXT>          ::=  "FFSExt=" <Extension>

<APRIORI>         ::=  "APRIORI=" <FvImageNameIdx>
                         ["," <FvImageNameIdx>]*

<FvImageNameIdx>  ::=  <FvImageName> ":" <PositiveInt>

<PositiveInt>     ::=  Integer value greater than 0

<EFN>             ::=  "EFI_FILE_NAME" "=" <Path> <Arch>
                         <FileSep> <Word>
```

Example

```
[files]
EFI_FILE_NAME = C:\Edk\Sample\Platform\Nt32\Build\IA32\2D2E62CF-9ECF-43b7-8219-
94E7FC713DFE-UsbKb.dxe
EFI_FILE_NAME = C:\Edk\Sample\Platform\Nt32\Build\IA32\A5C6D68B-E78A-4426-9278-
A8F0D9EB4D8F-UsbMassStorage.dxe
EFI_FILE_NAME = C:\Edk\Sample\Platform\Nt32\Build\IA32\2D2E62AA-9ECF-43b7-8219-
94E7FC713DFE-UsbMouse.dxe
EFI_FILE_NAME = C:\Edk\Sample\Platform\Nt32\Build\IA32\961578FE-B6B7-44c3-AF35-
6BC705CD2B1F-Fat.dxe
```

G.4 [Options] Section

Summary

Defines the `[options]` tag is found only in Firmware Volume INF files. This is an optional section.

Prototype

```
<options>     ::=  "[options]" <EOL>
                   <expression>+

<expression>  ::=  <Variable> "=" <Value> <EOL>

<Variable>    ::=  {"EFI_BASE_ADDRESS"} {"EFI_BLOCK_SIZE"}
                   {"EFI_FILE_NAME"} {"EFI_NUM_BLOCKS"}
                   {"EFI_SYM_FILE_NAME"} ("IA32_RST_BIN"}

<CName>       ::=  A valid C variable name
```

```
<VAL>          ::=  "0x" <HexDigit>{1,8}

<Value>        ::= {<String>} {<VAL>} {<Filename>}
```

Example

```
[options]
EFI_BASE_ADDRESS        = 0xFFD80000
EFI_FILE_NAME           = FvRecovery.fv
EFI_NUM_BLOCKS          = 0x28
EFI_BLOCK_SIZE          = 0x10000
```

Appendix H
HII - EFI Compliant Unicode Strings File Format

This appendix describes the EFI compliant format for the input Unicode string files that are parsed by the StrGather parser module of the AutoGen utility. The utility is versatile in that it supports multiple layouts and formats in the Unicode file. This versatility allows strings to be grouped either by language or string identifier, and strings may be demarcated with double quotes or by a new control character in the first column of a new line.

The token name, following the /string token must be in uppercase characters.

H.1 Control Character

The default control character is a forward-slash (/), with a double slash (//) indicating a comment to end-of-line. The control character can be changed at any time in the script using the "set control character" control sequence. For example to use a hash (#) instead of a slash:

/=#

H.1.1 Language Definition

Before any strings for a particular language can be defined, the language must be defined using the /langdef directive. For example, if strings for English and Spanish are to be defined in the Unicode string file, then the following statements must be present before any strings of the given language are defined.

```
/langdef eng "English"
/langdef spa "Spanish"
```

Once these directives have been specified, the shorthand "eng" or "spa" may be used to tag the languages of strings defined in the file.

H.2 String Definitions

The /string and /language directives are used to declare string identifiers and the corresponding strings for different languages. If a control character in column 1 is used to indicate the end of strings, then the Unicode string definitions for two strings for two different languages could take the form:

```
/string MY_STRING_1
/language eng
My first English string
/string MY_STRING_2
/language eng
My second English string
/string MY_STRING_1
/language spa
Mi primera secuencia inglesa
/string MY_STRING_2
/language spa
Mi segunda secuencia inglesa
```

If control characters are used to terminate strings (as shown above), and the string contains double-quotes as the first and last non-blank character on the first line of the input Unicode file, then the -uqs" command line option must be used.

By using quotes to mark the start and end of strings, a more columnar view is presented.

```
/string MY_STRING_1 /language eng "My first English string"
/string MY_STRING_2 /language eng "My second English string"
/string MY_STRING_1 /language spa "Mi primera secuencia inglesa"
/string MY_STRING_2 /language spa "Mi segunda secuencia inglesa"
```

Or to group strings by identifier rather than by language:

```
/string MY_STRING_1 /language eng "My first English string"
                    /language spa "Mi primera secuencia inglesa"
/string MY_STRING_2 /language eng "My second English string"
                    /language spa "Mi segunda secuencia inglesa"
```

Any of these formats may be used and all result in identical outputs of StrGather.

Longer strings can be split across lines, and the StrGather utility will combine them. For example, the following two string definitions are equivalent:

```
/string MY_STRING_1 /language eng "This string "
                                  "is split "

/string MY_STRING_1 /language eng "This string is split "
```

H.3 String Control Sequences

Several control sequences are also available within the actual strings. These sequences are converted to the appropriate binary encoding within the strings. These include the following:

\nbr

non-breaking code

\br

breaking code, not modified by StrGather

\narrow

display the following characters as narrow glyphs

\wide

display the following characters as wide glyphs

\n

insert carriage return and line feed

\r
insert carriage return

****
insert a single backslash

\"
insert a double-quote

Examples of how these would be used in the Unicode string file are listed below.

```
/string STR1 /language eng "This string is meant to be \n"
                           "split across two lines."
/string STR2 /language eng "This string has a \"quoted\" string"
/string STR3 /language eng "Path takes the form fs0:\\"
/string STR4 /language eng "Display \wide this text wide"
/string STR5 /language eng "Display \narrow this text narrow"
```

Appendix I
VS2005 Team Suite Performance Profile

This appendix provides the best known method for using Microsoft Visual Studio 2005 Team Suite to get performance data. It involves "porting" the EFI code to a Win32 console application and then using the VS Performance Wizard to figure out how to tune it.

I.1 Step 1 - Create a new project

On the Menu bar, select: File->New->Project..

In the Project types: frame select Visual C++ -> Win32

In the Templates: frame then select Win32 Console Application

- Give the project a name and a solution
- Accept the default settings from the wizard.

A <project name>.cpp file will be generated

```
<project name>.cpp
// Test.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
      return 0;
}
```

I.2 Step 2 - Update the project

You will need to update the new project to support reading in input files and writing data to an output file:

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>

void *
Malloc (
  int Size
  )
{
  return HeapAlloc (GetProcessHeap (), 0, Size);
}

int
_tmain(
  int     argc,
  _TCHAR* argv[]
  )
{
  HANDLE                      hFile;
  HANDLE                      hOutFile;
  DWORD                       Error;
  DWORD                       BytesRead;
  BY_HANDLE_FILE_INFORMATION  FileInfo;
  void                        *Buffer;
  int                         Status;
  UINT32                      DestinationSize;
  VOID                        *Destination;

  Status = 0;
  printf ("test %d\n", argc);
  if (argc <= 1) {
    return 0;
  }

  hFile = CreateFile (
            argv[1],
            GENERIC_READ,
            FILE_SHARE_READ,
            NULL,
            OPEN_EXISTING,
            0,
            0
            );
  if (hFile == INVALID_HANDLE_VALUE) {
    Error = GetLastError ();
    return Error;
  }

  if (!GetFileInformationByHandle (hFile, &FileInfo)) {
    Error = GetLastError ();
    return Error;
  }

  if (FileInfo.nFileSizeHigh != 0) {
    // Assume input file is less than 4GB in size
```

```
    return 0;
  }

  Buffer = Malloc (FileInfo.nFileSizeLow);

  if (!ReadFile (hFile, Buffer, FileInfo.nFileSizeLow, &BytesRead, NULL)) {
    Error = GetLastError ();
    return Error;
  }

  // Process File …
  // DestinationSize = …
  // Destination     = …

  // If a 2nd argument exists it is a file name to write data to
  if ((argc >= 3) && (Status == 0)) {
    hOutFile = CreateFile (
                argv[2],
                GENERIC_WRITE | GENERIC_READ,
                0,
                NULL,
                CREATE_ALWAYS,
                FILE_ATTRIBUTE_NORMAL,
                NULL
                );
    if (hOutFile != INVALID_HANDLE_VALUE) {
      if (!WriteFile (hOutFile, Destination, DestinationSize, &BytesRead, NULL))
{
        Error = GetLastError ();
      }

      CloseHandle (hOutFile);
    }
  }

  CloseHandle (hFile);
  return 0;
}
```

I.2.1 To pass an argument in to the console application

Do the following:

1. Update the <project name> Property Pages:
2. Right click on the <project name> in the Solution Explorer pain
3. Select preferences
4. In the configurations: window select All Configurations
5. In the left hand pain select Configuration Properties->Debugging
6. Under Command Arguments type in the command line. In my example the input file is compress and the output file is decompress.out

In this example compress is the EDK II NT32 FV (2.5MB) compressed to 707K.

So decompress.out should be 2.5MB NT32 FV.

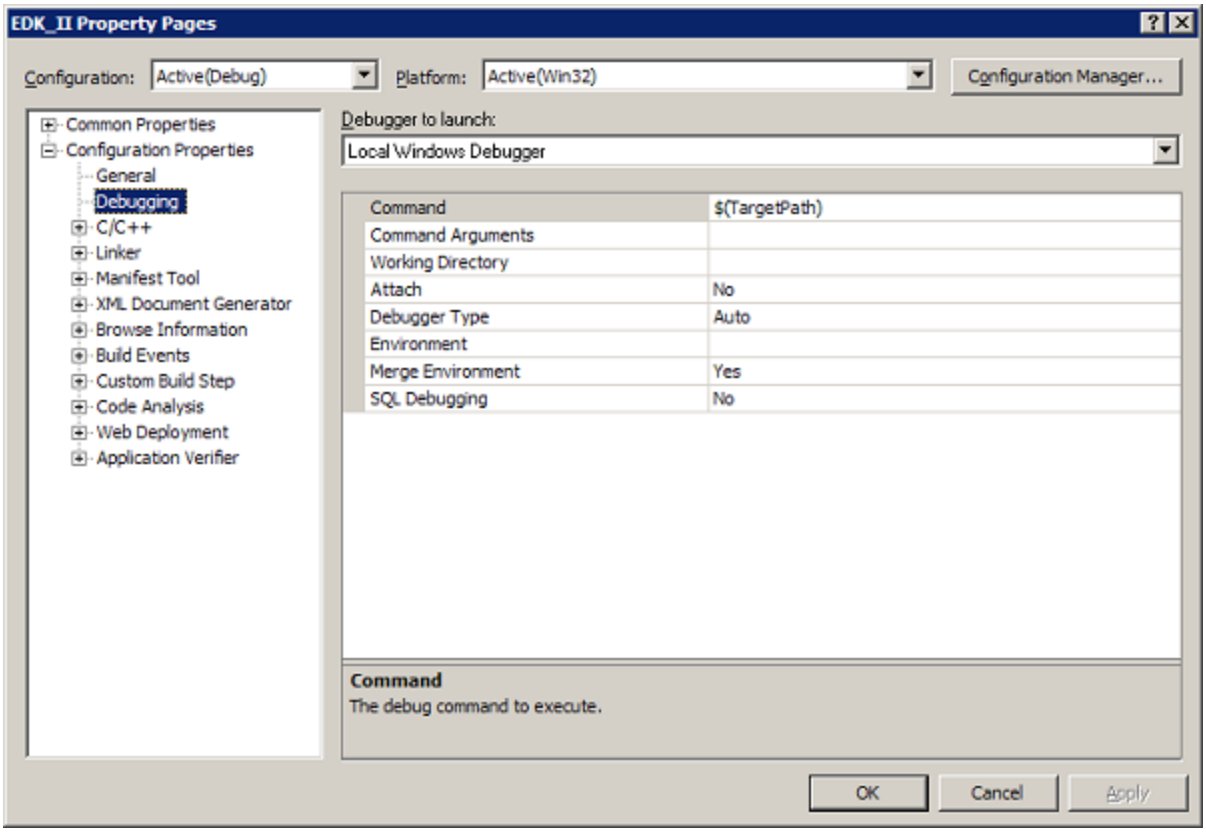


Figure 24. VS2005 Property Page

This example required the EDK II Decompress Lib be ported into this environment as follows:

1. Add EDK II EFI type definitions to get the EFI code to compile.

```
//
// Map EFI types
//
typedef unsigned __int64    UINT64;
typedef __int64             INT64;
typedef unsigned __int32    UINT32;
typedef __int32             INT32;
typedef unsigned short      UINT16;
typedef unsigned short      CHAR16;
typedef short               INT16;
typedef unsigned char       BOOLEAN;
typedef unsigned char       UINT8;
typedef char                CHAR8;

#define UINT8_MAX 0xff
```

2. Convert EFI_STATUS/RETURN_STATUS to int and removed #defines for return values to make it easier for the code to compile.
3. Glue in the EFI code into _tmain()

```
  // Process File
  Status = UefiDecompressGetInfo (Buffer, FileInfo.nFileSizeLow,
&DestinationSize, &ScratchSize);
  if (Status == 0) {
    Destination = Malloc (DestinationSize);
    Scratch = Malloc (ScratchSize);

    if ((Scratch != NULL) && (Destination != NULL)) {
      Status = UefiTianoDecompress (Buffer, Destination, Scratch, 2);
      if (Status != 0) {
        printf ("Decompress Failed");
      }
    }
  }
```

I.2.2 Step 3 Run the Performance Wizard

1. Tools->Performance Tools->Performance Wizard…
2. Make sure your project is selected and hit Next
3. When you are asked what method of profiling would like to use select Instrumentation.
   a The default is Sampling so you must change this
4. Then type Finish
5. A Performance Explorer pain should show up.
6. Right click on you project name and select Launch
   a This should rebuild your application with performance infrastructure.
   b Under Reports you should see a <Project Name>[date].vsp file that contains the info

Make sure you profile in the Release build and not the Debug build for best results.

The following is an example of the output you should see.

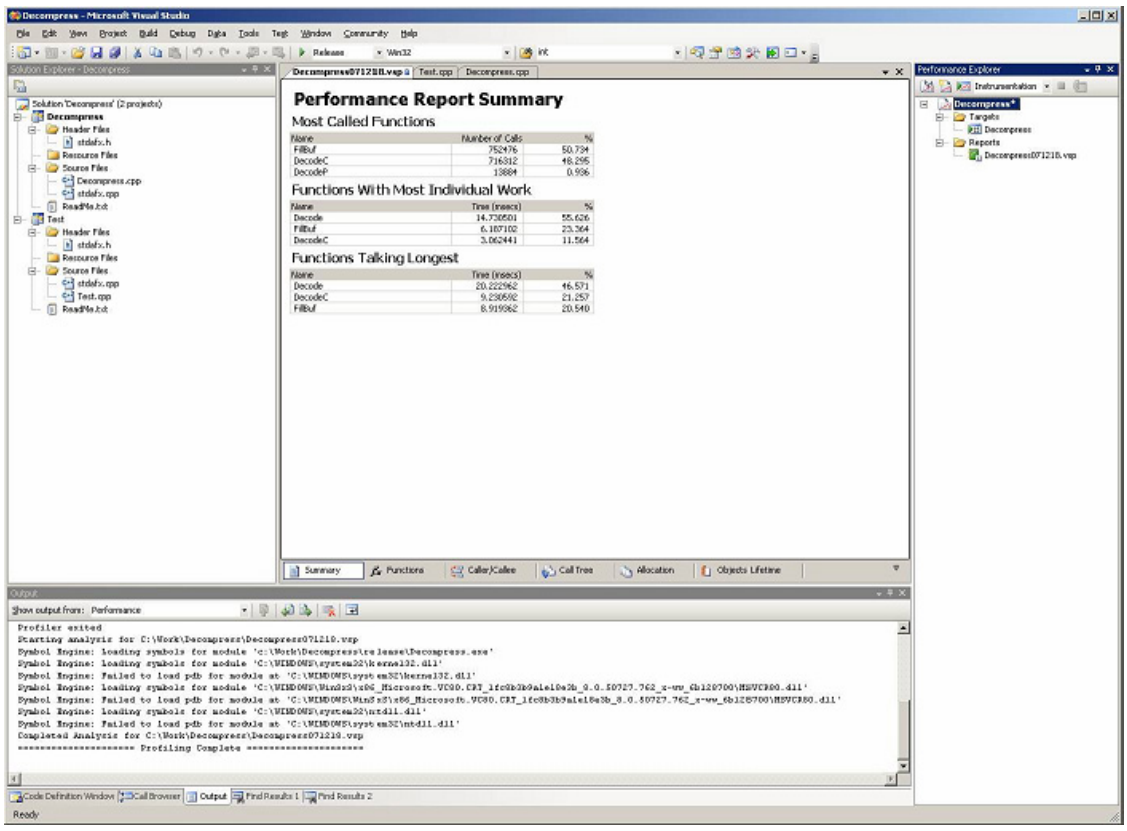


Figure 25. VS2005 Performance Summary

From the summary, it appears that Decode() must have a very hot loop in it. DecodeC and FillBuf are very simple, but they are called so many times a very small improvement will be multiplied by 100,000.

Expanding the call tree view can be very useful.

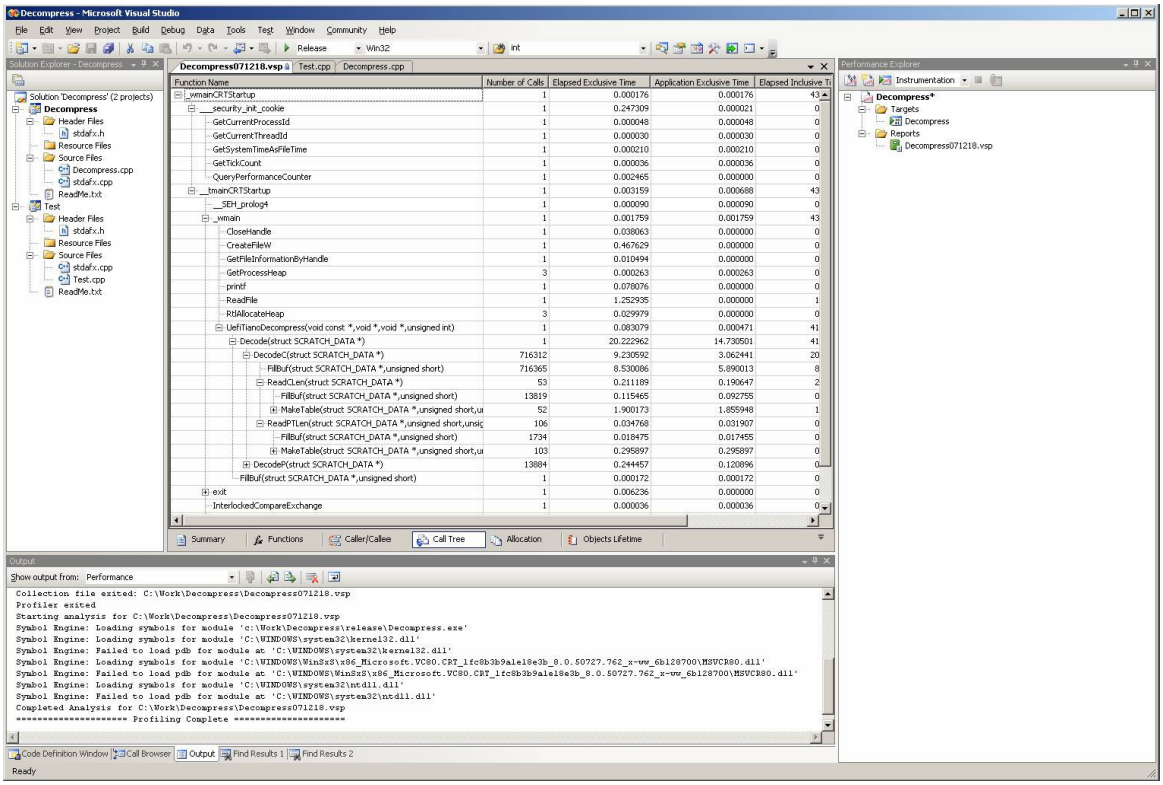

Decompress - Microsoft Visual Studio
Function Name
Number of Calls
Elapsed Exclusive Time
Application Exclusive Time
Summary
Functions
Caller/Callee
Call Tree
Allocation
Objects Lifetime
Output
Collection file exited: C:\Work\Decompress\Decompress071218.vsp
Profiler exited
Starting analysis for C:\Work\Decompress\Decompress071218.vsp
Symbol Engine: Loading symbols for module 'c:\Work\Decompress\release\Decompress.exe'
Symbol Engine: Loading symbols for module 'C:\WINDOWS\system32\kernel32.dll'
Symbol Engine: Failed to load pdb for module at 'C:\WINDOWS\system32\kernel32.dll'
Symbol Engine: Loading symbols for module 'C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\MSVCR80.dll'
Symbol Engine: Failed to load pdb for module at 'C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\MSVCR80.dll'
Symbol Engine: Loading symbols for module 'C:\WINDOWS\system32\ntdll.dll'
Symbol Engine: Failed to load pdb for module at 'C:\WINDOWS\system32\ntdll.dll'
Completed Analysis for C:\Work\Decompress\Decompress071218.vsp
Ready

Figure 26. VS2005 Call Tree View

Definition of terms http://msdn2.microsoft.com/en-us/library/ms242753(VS.80).aspx

Appendix J
HII - UEFI Compliant Unicode Strings File Format

This append describes the UEFI compliant format for the input Unicode string files (and strings within header files) that are parsed by the StrGather parser module of the AutoGen utility. The utility is versatile in that it supports multiple layouts and formats in the Unicode file. This versatility allows strings to be grouped either by language or string identifier, and strings may be demarcated with double quotes or by a new control character in the first column of a new line.

Comments may appear anywhere within the string file, however processing of the line will stop at a comment that is appended to a line.

Blank lines terminate processing of a multi-line string, as does the end of file. Additionally, a control character (default is "/" forward-slash) will also terminate the end of a string.

UCS-2 compliant files begin with a binary character, 0xFEFF (big-endian). These file may be created either by using a text editor program that will do conversions or by creating an ASCII text file and processing the file to create the UCS-2 file.

Unicode strings defined in a C file are quoted ASCII text, prefixed with a "L" character. These types of strings may also be defined in meta-data files.

```
L"This string must be converted to UCS-2 format"
```

Because the string token name in UNI file will be generated into AutoGen header file as #define MACRO name. EDKII coding style document requires using all capital letters for both #define and typedef declarations. So, string token name in UNI file must also be uppercase unicode characters.

Prototype

```
<StringFileFormat>  ::=  [<CommentLine>]*
                         <LanguageDefs>
                         <Content>+

<LanguageDefs>      ::=  <CtrlChar> "langdef" <LangCode> <QuotedWord> <EOL>

<Content>           ::=  {<CommentLine>} {<BlankLine>} {<UnicodeLines>}
                         {<ControlRefactor>} {<LanguageDefs>} {<SecurityLines>}

<CommentLine>       ::=  L"//" <Text> <EOL>

<Text>              ::=  [<Whitespaces>]* <Chars>+ [<Text>]*

<Chars>             ::=  (0x0001-0xF6FF)

<DefaultCtrlChar>   ::=  L"/"

<ControlRefactor>   ::=  <CtrlChar> L"=" <NewCtrlChar> <EOL>

<UnicodeLines>      ::=  <Token> [<CtrlChar> L"language" <LangCode> <String>]+

<SecurityLines>     ::=  L"security" L"(" <CFormatGUID> L")" <EOL>

<CFormatGUID>       ::=  L"{ 0x" <HexDigitU>{8} L", 0x" <HexDigitU>{4}
                         L", 0x" <HexDigitU>{4} L", { 0x" <HexDigitU>{2}
                         L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                         L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                         L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                         L", 0x" <HexDigitU>{2} L" }}"

<HexDigitU>         ::=  (\ua-\uf\uA-\uF\u0-\u9)

<Token>             ::=  <CtrlChar> L"string" (\uA-\uZ)(\uA-\uZ\u0-\u9\u_){0,}

<LangCode>          ::=  {<ISO639part1>} {<ISO639part2>} {<RFC3066>}

<ISO639part1>       ::=  (\ua-\uz\uA-\uZ){2}

<ISO639part2>       ::=  (\ua-\uz\uA-\uZ){3}

<RFC3066>           ::=  (\ua-\uz\uA-\uZ){2,8} [L"-" (\ua-\uz\uA-\uZ\u0-\u9){2,8}]

<QuotedWord>        ::=  {L"'" L<Word> L"'"} {L'"' L<Word> L'"'}

<Word>              ::=  (NonWhitespaceChar)(NonWhitespaceChar)*

<String>            ::=  [<Text>] [<Attributes>] [<CtrlCode>] [<EOL>]
                         [<String>] <EOL>

<Attributes>        ::=  <StartAttribute> <String> [<StopAttribute>]

<StartAttribute>    ::=  <AttrCtrlChar> <FontAttr>
```

```
<AttrCtrlChar>      ::=  L"\"

<StopAttribute>     ::=  <AttrCtrlChar> L"end" <FontAttr>

<FontAttr>          ::=  {<SimpleAttrs>} {<StandardAttrs>}

<SimpleAttrs>       ::=  {L"narrow">} {<L"wide">}

<StandardAttrs>     ::=  {L"normal"} {L"bold">} {<L"italic">} {<L"emboss">}
                         {L"shadow"} {L"underline"} {L"dblunder"}

<CtrlCode>          ::=  {L"\n"} {L"\f"} {L"\r"} {L"\p"} {L"\ospace"}
                         {L"\enquad"} {L"\emquad"} {L"\ensp"} {L"\emsp"}
                         {L"\em3sp"} {L"\em4sp"} {L"\em6sp"} {L"\usp"}
                         {L"\tsp"} {L"\hsp"} {L"\msp"} {L"\!bsp"} {L"\!nbsp"}
                         {L"\zsp"} {L"\ah"} {L"\hy"} {L"\df"} {L"\den"}
                         {L"\dem"} {L"\!bh"} {L"\g"} {L"\osp"} {L"\k"}

<EscChar>           ::=  L"\"
```

Definitions

LanguageCodes

If a 2 character ISO 639-1 language code exists along with a 3 character ISO 639-2 language code, then the ISO 639-1 language code must be used. When a language has no ISO 639-1 2-character code, and the ISO 639-2/T (Terminology) code and the ISO 639-2/B (Bibliographic) code differ, you MUST use the Terminology code. When a language has both an IANA-registered tag (i-something) and a tag derived from an ISO registered code, you MUST use the ISO tag.

EscChar

In order to include some standard characters, such as the "\" back-slash character within a string, the character must be prefixed with the escape character. Characters that may require a prefixed escape character include the following, back slash "\" character, single-quote "'" character, double-quote '"' character and the forward slash "/" character. The back slash always requires the escape character.

StandardAttrs

The standard font attribute, "normal" was not defined in the *UEFI 2.1 Specification*, however it has been proposed and is included here. Additional attributes defined in the *UEFI 2.1 Specification*, such as double underline (dblunder), did not have the double-byte encoding for the character mapping, however recommendations have been given for these characters (see Table 29 below).

Table 29. HII Double-Byte Encoding Map

String	Double-Byte Encoding	String	Double-Byte Encoding
\bold	0xF620	\endbold	0xF621
\italic	0xF622	\enditalic	0xF623
\underline	0xF624	\endunderline	0xF625
\dblunder	0xF62A	\enddblunder	0xF62B5
\emboss	0xF6265	\endemboss	0xF6275
\shadow	0xF6285	\endshadow	0xF6295
\n (newline)	0x2028	\f (formfeed)	0x000C
\r (carriage return)	0x000D	\p (paragraph separator)	0x2029
\ospace (ogham space mark)	0x1680	\enquad	0x2000
\emquad	0x2001	\ensp (en space)	0x2002
\emsp	0x2003	\em3sp (three-per-em space)	0x2004
\em4sp	0x2005	\em6sp	0x2006
\usp (punctuation space)	0x2008	\tsp (thin space)	0x2009
\hsp (hair space)	0x200A	\msp (medium math space)	0x205F
\!bsp (no-break space)	0x00A0	\!nbsp (narrow no-break space)	0x0202F
\zsp (zero width space)	0x200B	\ah (Armenian hyphen)	0x058A
\hy (hyphen)	0x2010	\df (figure dash)	0x2012
\den (en dash)	0x2013	\dem (em dash)	0x2014
\!bh (non-breaking hyphen)	0x2011	\g (Tibetan mark intersyllabic tsheg)	0x0F0B
\osp (Ethiopic wordspace)	0x1361	\k (Khmer sign bariyoosan)	0x17D5

Appendix K
Module Types

Table 30. EDK II Module Types

MODULE_TYPE	Supported Architecture Types	Description
BASE	Any	Modules or Libraries can be ported to any execution environment. This module type is intended to be used by silicon module developers to produce source code that is not tied to any specific execution environment.
SEC	Any	Modules of this type are designed to start execution at the reset vector of a CPU. They are responsible for preparing the platform for the PEI phase.
PEI_CORE	Any	This module type is used by PEI Core implementations that are compliant with the PI 1.2 Specification.
PEIM	Any	This module type is used by PEIMs that are compliant with PI 1.2.
DXE_CORE	Any	This module type is used by DXE Core implementations that are compliant with the PI 1.2 Specification.
DXE_DRIVER	Any	This module type is used by DXE Drivers that are compliant with the PI 1.2 Specification.
DXE_RUNTIME_DRIVER	Any	This module type is used by DXE Drivers that are compliant to the PI 1.2 Specification. These modules execute in both boot services and runtime services environments.
DXE_SAL_DRIVER	IPF	This module type is used by DXE Drivers that can be called in physical mode before SetVirtualAddressMap() is called and either physical mode or virtual mode after SetVirtualAddressMap() has been called. This module type is only available for IPF processor types.
DXE_SMM_DRIVER	IA32, X64	This module type is used by DXE Drivers that are loaded into SMRAM.
SMM_CORE	Any	This is the SMM core.
UEFI_DRIVER	Any	This module type is used by UEFI Drivers that are compliant with the EFI 1.10 and UEFI 2.0 specifications. These modules provide services in the boot services execution environment. UEFI Drivers that return EFI_SUCCESS are not unloaded from memory. UEFI Drivers that return an error are unloaded from memory.
UEFI_APPLICATION	Any	This module type is used by UEFI Applications that are compliant with the EFI 1.10 and EFI 2.0 specifications. UEFI Applications are always unloaded when they exit.